On Making Directed Graphs Eulerian

Manuel Sorge

Diplomarbeit

zur Erlangung des akademischen Grades
Diplom-Informatiker

Friedrich-Schiller-Universität Jena
Institut für Informatik
Theoretische Informatik I / Komplexitätstheorie

Betreuung:	Überarbeitete Version
Dipl.-Inf. René van Bevern	Eingereicht von Manuel Sorge,
Prof. Dr. Rolf Niedermeier	geb. am 28.04.1986 in Zittau.
Dipl.-Inf. Mathias Weller	Jena, den 21. Februar 2011

Zusammenfassung. Einen gerichteten Graphen nennt man *Eulersch*, wenn er eine Tour enthält, die jede gerichtete Kante genau einmal besucht. Wir untersuchen das Problem EULERIAN EXTENSION (EE) in dem ein gerichteter Multigraph G und eine Gewichtsfunktion gegeben ist, und gefragt wird, ob G durch Hinzufügen gerichteter Kanten, deren Gesamtgewicht einen Grenzwert nicht überschreitet, Eulersch gemacht werden kann. Dieses Problem ist motiviert durch Anwendungen im Erstellen von Fahrplänen für Fahrzeuge und Abfolgeplänen für Fließbandarbeit. Allerdings ist das Problem EE NP-schwer; deshalb analysieren wir es mit Hilfe von Parametrisierter Komplexität. Die Parametrisierte Komplexität eines Problems hängt nicht nur von der Eingabelänge, sondern auch von anderen Eigenschaften der Eingabe ab. Diese Eigenschaften nennt man "Parameter". Dorn et al. [10] zeigten, dass EE in $O(4^k n^4)$ Zeit gelöst werden kann. Hier bezeichnet k den Parameter "Anzahl der gerichteten Kanten, die hinzugefügt werden müssen". In dieser Arbeit analysieren wir EE mit den (kleineren) Parametern "Anzahl c der verbunden Komponenten im Eingabegraph" und "Summe b aller $\mathrm{indeg}(v) - \mathrm{outdeg}(v)$ über alle Knoten v für die dieser Wert positiv ist". Wir zeigen dass es einen Lösungsalgorithmus für EE gibt, dessen Laufzeit den Term $4^{c \log(bc^2)}$ als einzigen superpolynomiellen Term beinhaltet. Um diesen Algorithmus zu erhalten, machen wir mehrere Beobachtungen über die Mengen gerichteter Kanten, die dem Eingabegraph hinzugefügt werden müssen, um ihn Eulersch zu machen. Aufbauend auf diesen Beobachtungen geben wir außerdem eine Reformulierung von EE in einem Matchingkontext. Diese Matchingformulierung könnte ein bedeutendes Werkzeug sein, um zu klären, ob EE in Laufzeit gelöst werden kann, deren superpolynomieller Anteil nur von c abhängt. Außerdem betrachten wir Vorverarbeitungsalgorithmen polynomieller Laufzeit für EE, und zeigen, dass diese keine Instanzen erzeugen können, deren Größe polynomiell nur von einem der Parameter b, c, k abhängt, es sei denn coNP $\subseteq$ NP/poly.

Abstract. A directed graph is called *Eulerian*, if it contains a tour that traverses every arc in the graph exactly once. We study the problem of EULERIAN EXTENSION (EE) where a directed multigraph G and a weight function is given and it is asked whether G can be made Eulerian by adding arcs whose total weight does not exceed a given threshold. This problem is motivated through applications in vehicle routing and flowshop scheduling. However, EE is NP-hard and thus we use the parameterized complexity framework to analyze it. In parameterized complexity, the running time of algorithms is considered not only with respect to input length, but also with respect to other properties of the input—called "parameters". Dorn et al. [10] proved that EE can be solved in $\mathrm{O}(4^k n^4)$ time, where k denotes the parameter "number of arcs that have to be added". In this thesis, we analyze EE with respect to the (smaller) parameters "number c of connected components in the input graph" and "sum b over $\mathrm{indeg}(v) - \mathrm{outdeg}(v)$ for all vertices v in the input graph where this value is positive". We prove that there is an algorithm for EE whose running time is polynomial except for the term $4^{c \log(bc^2)}$. To obtain this result, we make several observations about the sets of arcs that have to be added to the input graph in order to make it Eulerian. We build upon these observations to restate Eulerian extension in a matching context. This matching formulation of EE might be an important tool to solve the question of whether EE can be solved within running time whose superpolynomial part depends only on c. We also consider polynomial time preprocessing routines for EE and show that these routines cannot yield instances whose size depends polynomially only on either of the parameters b, c, k unless $\mathrm{coNP} \subseteq \mathrm{NP/poly}$.

Contents

Chapter 1

Introduction

The notion of Eulerian graphs dates back to Leonhard Euler. In 1735, he solved the question of whether there is a tour through the city of Königsberg such that every bridge is crossed once and only once [14]. The term "Eulerian tour" has been coined for such a tour. At Euler's time, the city of Königsberg had seven bridges across the river Pregel and it turned out that an Eulerian tour did not exist. Later, in the nineteenth century, a railway bridge has been built, making such a tour feasible [30].

In this thesis, we study the problem where, given a city, it is asked what is a minimum-cardinality set of bridges that have to be built such that the city allows for an Eulerian tour? Typically, one aims for minimizing the costs for bridge-building. Thus, we mainly focus on a weighted version of this problem. Also, instead of ordinary bridges, we consider bridges that only allow traffic in one way. We call the problem of making a city allow for an Eulerian tour by building one-way bridges, weighted according to a cost function, EULERIAN EXTENSION.

The problem of EULERIAN EXTENSION is also well-motivated through wholly different approaches. Recently, it has been shown by Höhn et al. [21] that some sequencing problems can be solved with EULERIAN EXTENSION. An example for such a sequencing problem is "no-wait flowshop", where a schedule of jobs is sought, each processing on a fixed succession of machines, such that no waiting time occurs for any job between the processing on two subsequent machines. Such problems arise, for instance, in steel production [20].

Another problem that has strong ties to EULERIAN EXTENSION is RURAL POSTMAN. There, a postman's tour in a city is sought such that a given subset of streets in the city is serviced. Dorn et al. [10] have proven that RURAL POSTMAN is equivalent to EULERIAN EXTENSION. RURAL POSTMAN can be applied, for example, in routing of snow plowing vehicles [13].

Unfortunately, EULERIAN EXTENSION is NP-hard, and thus it is likely not to be solvable within time polynomial in the input size. Thus, we analyze EULERIAN EXTENSION using the parameterized-complexity framework. That is, we consider running times not only depending on the input size, but also depending on other properties of the input—called parameters. A problem is called fixed-parameter tractable for a specific parameter if there is an algorithm whose exponential running-time portion depends only on this parameter. For instance, Dorn et al. [10] have proven that EULERIAN EXTENSION is fixed-parameter tractable

with respect to the parameter k = "number bridges that have to be built". They have shown that EULERIAN EXTENSION can be solved with an algorithm that has running time $O(4^k n^4)$.

In this work, we consider smaller parameters for EULERIAN EXTENSION. In particular, we look at the number of "islands" the input city consists of and a more technical parameter which we introduce later. By islands we mean districts in the city that are completely cut off from other districts and cannot be reached via bridges. On the positive side, despite EULERIAN EXTENSION being NP-hard, we are able to derive algorithms for this problem whose exponential running time portion depends only on these two, presumably small, parameters. On the negative side, we observe that preprocessing EULERIAN EXTENSION such that the size of the resulting instances is bounded by polynomials in these parameters likely is not possible.

Our work is organized as follows: In Section 1.1, we gather a common knowledge base by recapitulating basic notions of graph theory and parameterized algorithmics. Section 1.2 treats previous work on EULERIAN EXTENSION and the related problem of RURAL POSTMAN. There we also give an NP-hardness proof for EULERIAN EXTENSION for illustrative purposes. In Chapter 2, constituting the main part of this work, we consider structural properties of EULERIAN EXTENSION and derive an efficient algorithm. We also restate the problem in a matching context. This matching formulation might become a stepping stone to solve the question of whether EULERIAN EXTENSION is fixed-parameter tractable with respect to the parameter "number of islands in the city" we have sketched above. Chapter 3 contains our considerations with regard to preprocessing routines for EULERIAN EXTENSION. There, it is shown that no polynomial-time data reduction rules exist that reduce an instance to a polynomial-size problem kernel unless coNP $\subseteq$ NP/poly. Finally in Chapter 4, we give a brief summary of our results and give directions for further research.

1.1 Preliminaries

We assume the reader to be familiar with the basics of set theory, logic, algorithm analysis, and computational complexity. They are covered, for example, by Cormen et al. [7] and Arora and Barak [1].

1.1.1 Parameterized Algorithmics

Many problems of practical importance are NP-hard. Thus, it is widely believed that for these problems there is no algorithm whose running time is bounded by a polynomial in the input size. However, in practice sometimes the phenomenon can be observed, that instances of such problems are indeed solvable within reasonable time. The reason for this is that some algorithms can be analyzed such that their super-polynomial running time portion depends only on some property of the input instances. If such a property, called "parameter", is not dependent on the input size and if it is "small" in practical instances, then it becomes feasible to solve even big instances of NP-hard problems. The notion of parameters also gives rise to a method measuring the effectiveness of data reduction rules: If the rules reduce an instance such that its size is bounded by

polynomials depending only on the parameter, we may assume that the reduction rules perform well in practice.

The design of algorithms exploiting small parameters is treated in Niedermeier [27]. However, it is likely that not every problem admits such an algorithm for every possible parameter. In this regard, complexity-theoretic approaches are presented in Downey and Fellows [11] and Flum and Grohe [16]. We use the parameterized complexity framework in our analysis of EULERIAN EXTENSION and give some basic definitions of parameterized algorithms and complexity here.

Problems and Parameterizations. Let Σ be an alphabet. A *parameterization* is a polynomial-time computable function $\kappa : \Sigma^* \to \mathbb{N}$. A *parameterized problem* over Σ is a tuple (Q, κ) where $Q \subseteq \Sigma^*$ and κ is a parameterization. For an instance $I \in \Sigma^*$ of a parameterized problem (Q, κ) we also call $\kappa(I)$ the *parameter*. A parameterized problem (Q, κ) is called *fixed-parameter tractable* with respect to κ, if there is an algorithm that, given an instance $I \in \Sigma^*$, decides whether $I \in Q$ in time at most $f(\kappa(I)) \cdot p(\text{length}(I))$. Here, $f : \mathbb{N} \to \mathbb{N}$ is a computable function and p is a polynomial.

Search Tree Algorithms. A straightforward way to prove a problem fixed-parameter tractable is to employ search tree algorithms. A *search tree algorithm* recursively divides an instance of a problem into a number of new instances. It divides the instances, until they are solvable within polynomial time. If both the recursion depth of the algorithm and the number of new instances generated from each instance is bounded by the parameter, then we obtain an algorithm whose superpolynomial time-portion is bounded by some function depending only on the parameter. Thus, the algorithm is a witness to the fixed-parameter tractability of the problem. Since search tree algorithms often terminate early, are easily parallelized and often allow for many performance tweaks, they often form relevant practical algorithms for NP-hard problems.

Problem Kernels. Problem kernels are often used to give a guarantee of efficiency for data reduction rules. Let (Q, κ) be a parameterized problem. A *reduction rule* for (Q, κ) is a mapping $r : \Sigma^* \to \Sigma^*$ such that
    (1) r is polynomial-time computable,
    (2) for every $I \in \Sigma^*$ it holds that $I \in Q$ if and only if $r(I) \in Q$, and
    (3) $\kappa(r(I)) \leq \kappa(I)$.
Statement (2) is called the *correctness* of the rule. A reduction to a *problem kernel* for (Q, κ) is a reduction rule $r : \Sigma^* \to \Sigma^*$ such that for every $I \in \Sigma^*$ it holds that $\text{length}(r(I)) \leq h(\kappa(I))$, where h is a computable function. If h is a polynomial, then we call r a reduction to a *polynomial-size problem kernel*.

Parameterized Reductions. Let (Q, κ), and (Q', κ') be two parameterized problems. A *parameterized many-one reduction* from (Q, κ) to (Q', κ') is a mapping $r : \Sigma^* \to \Sigma^*$ such that for every $I \in \Sigma^*$
    (1) $r(I)$ is computable in time at most $f(\kappa(I)) \cdot p(\text{length}(I))$, where $f : \mathbb{N} \to \mathbb{N}$ is a computable function and p a polynomial,
    (2) $r(I) \in Q'$ if and only if $I \in Q$, and
    (3) there is a computable function $g : \mathbb{N} \to \mathbb{N}$ such that $\kappa'(r(I)) \leq g(\kappa(I))$.

Statement (2) is called the *correctness* of the reduction. If the function f is a polynomial in statement (1) we call r a *parameterized polynomial-time many-one reduction.* If the function g in statement (3) is a polynomial, we call r a *polynomial-parameter many-one reduction.* A *parameterized Turing reduction* from (Q, κ) to (Q', κ') is an algorithm with an oracle to Q' that, given an instance $I \in \Sigma^*$,

(1) decides whether $I \in Q$,

(2) has running time at most $f(\kappa(I)) \cdot p(\text{length}(I))$, where $f : \mathbb{N} \to \mathbb{N}$ is a computable function and p a polynomial, and

(3) there is a computable function $g : \mathbb{N} \to \mathbb{N}$ such that for every oracle query $I' \in \Sigma^*$ posed by the algorithm it holds that $\kappa(I') \leq g(\kappa(I))$.

Parameterized Complexity. It is assumed that not all parameterized problems are fixed-parameter tractable. To distinguish the various degrees of (in-)tractability, there is a multitude of complexity classes. We briefly introduce the classes

$$\text{FPT} \subseteq \text{W}[1] \subseteq \text{W}[2] \subseteq \ldots \subseteq \text{W}[P] \subseteq \text{XP}.$$

For a brief introduction to parameterized intractability and the W-hierarchy see Chen and Meng [6], for comprehensive works see Downey and Fellows [11] or Flum and Grohe [16].

The class FPT contains all parameterized problems that are fixed-parameter tractable. The class $\text{W}[t]$, $t \in \mathbb{N}$ contains all problems that are parameterized many-one reducible to the satisfiability problem for circuits with depth at most t and an AND output gate, parameterized by the weight of the sought truth assignment—that is the number of variables that are assigned true. The class $\text{W}[P]$ contains all parameterized problems (Q, κ) such that it can be decided whether a word $I \in \Sigma^*$ is contained in Q in at most $f(\kappa(I)) \cdot p(\text{length}(I))$ time, using a Turing machine that makes at most $h(\kappa(I)) \cdot \log(\text{length}(I))$ nondeterministic steps. Here, $f, h : \mathbb{N} \to \mathbb{N}$ are computable functions and p is a polynomial. The class XP contains all parameterized problems (Q, κ) for which there is a computable function f such that it can be decided whether $I \in \Sigma^*$ is contained in Q in time at most $\text{length}(I)^{f(\kappa(I))} + f(\kappa(I))$.

A parameterized problem (Q, κ) is assumed to be fixed-parameter intractable, if it is hard for the class of problems W[1]. That is, all parameterized problems in W[1] are parameterized many-one reducible to (Q, κ). Hardness for W[1] can be shown via a parameterized many-one reduction from a W[1]-hard parameterized problem. Such a problem is, for instance, INDEPENDENT SET parameterized by the size of the sought independent set k:

INDEPENDENT SET
Input: A graph $G = (V, E)$ and an integer k.
Question: Is there a vertex subset $S \subseteq V$ such that $k \leq |S|$ and $G[S]$ contains no edges?

1.1.2 Graphs

We now recapitulate some basic notions of graph theory. We oriented our definitions towards the ones given by Bang-Jensen and Gutin [2]. Other books on graph theory include Diestel [8] and West [29]. We also give some non-canonical

definitions for notions that we frequently use; especially in the "Connectivity" and "Degree and Balance" paragraphs below.

A *directed multigraph* G is a tuple (V, A), where V is a set, A is a multiset and for every $a \in A : a = (u, v) \in V \times V \wedge u \neq v$.[1] We sometimes denote V by $V(G)$ and A by $A(G)$. Elements in V are called the *vertices* of G and elements in A are called the *arcs* of G. We denote $|V|$ by n and $|A|$ by m where it is appropriate. A vertex $v \in V$ and an arc $(u, w) \in A$ are called *incident* if $u = v$ or $w = v$. Two vertices $u, v \in V$ are called *adjacent* or *neighbors* if there is an arc $a \in A$ such that u, v and a are incident. Let B be an arc set. We define the directed multigraph $G + B$ as $(V, A \cup B)$.

Subgraphs. Any directed multigraph (V', A') such that $V' \subseteq V$ and $A' \subseteq A$ is called a *subgraph* of G. Let $V' \subseteq V$ be a vertex set. The graph $G[V'] := (V', B)$ where $B = A \cap V' \times V'$ is called the *vertex-induced subgraph* of G with respect to V'. Let $A' \subseteq A$ be an arc-set. The graph $G\langle A' \rangle := (W, A')$ where $W = \{v \in V : \exists a \in A' : v$ and a are incident$\}$ is called the *arc-induced subgraph* of G with respect to A'.

Walks, Trails, and Paths. A *walk* is an alternating sequence

$$v_1, a_1, v_2, \ldots, v_{k-1}, a_{k-1}, v_k$$

of vertices $v_i \in V$ and arcs $a_j \in A$ such that $a_j = (v_j, v_{j+1})$ for all $1 \leq j \leq k - 1$. A *subwalk* of a walk w is a consecutive subsequence of w beginning and ending with a vertex. We say that a walk *traverses* a vertex v (an arc a) if the vertex v (the arc a) is contained in the corresponding sequence. We say that a walk w traverses a vertex set (arc set), if all vertices (arcs) in the set are traversed by w. The *length* of a walk is the number of arcs it traverses. The first vertex of a walk w is called the *initial* vertex and the last vertex is called the *terminal* vertex. The initial and terminal vertices are also called the *endpoints* of w. We say that a walk is *closed* if its initial and terminal vertices are equal. A *trail* in the graph G is a walk that traverses every arc of G at most once. A *path* in the graph G is a trail that traverses every vertex of G at most once. A closed trail that traverses every vertex of G at most once except for its initial and terminal vertices is called a *cycle*. We sometimes abuse notation and identify walks with their corresponding arc sets or their arc-induced subgraphs.

Graphs and Orientations. A *multigraph* G is a tuple (V, E), where V is a set, E is a multiset and for every $e \in E : e \subseteq V \wedge |e| = 2$. We sometimes denote V by $V(G)$ and E by $E(G)$. Vertices, edges, incidence, (vertex- or edge-induced) subgraphs, walks, trails and paths are defined analogously to the definitions for multigraphs. *Directed graphs* (digraphs for short) and *graphs* are the special cases of multigraphs that comprise arc or edge sets instead of multisets, respectively. A (directed) graph is called *complete* if it contains all possible edges (arcs). Let $G = (V, E)$ be a (directed) graph and let $G' = (V, E')$ be a complete (directed) graph. The *complement graph* of G is the graph $G = (V, \overline{E})$, where $\overline{E} = E' \setminus E$.

---

[1] We exclude "self-loops" $(v, v) \in A$ here, because they are not meaningful in the context of Eulerian extensions.
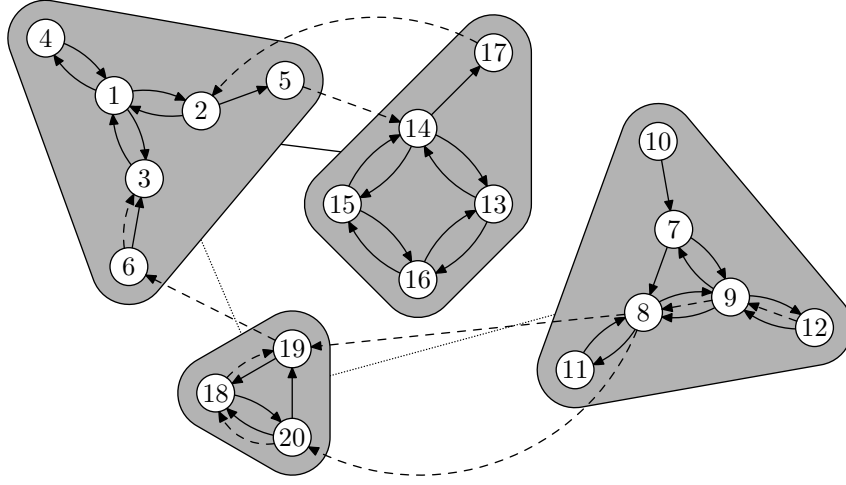
Figure 1.1: A directed graph G (vertices 1 through 20, solid arcs), and its components (encircled and shaded in gray). Furthermore, a number of trails is shown that traverse vertices of G (dashed arcs). The mapping $\mathbb{C}_G()$ maps both the trails traversing the vertices $5, 14$ and $17, 2$, respectively, to trail in $\mathbb{C}_G$ that is represented by the solid line. The trails traversing the vertices $12, 9, 8, 19, 6$ and $8, 20, 18, 19, 6, 3$, respectively, both map to the trail represented by the dotted lines.

A directed multigraph is said to be an *orientation* of a multigraph G if it can be obtained from G by substituting every edge $\{u, v\}$ by either the arc (u, v) or the arc (v, u). The *underlying multigraph* of a directed multigraph G is the uniquely determined multigraph G' such that G is an orientation of G'.

Drawing Graphs. We draw a directed multigraph by drawing circles for vertices, sometimes drawing their names inside the circle, and by drawing arrows with the head at the circle corresponding to the vertex u for arcs (v, u). Multigraphs are drawn by drawing circles for vertices and by drawing lines between the corresponding circles for edges.

Connectivity. A (directed) multigraph G is said to be *connected* if for every pair of vertices $u, v \in V(G)$ there is a path with the endpoints u, v in (the underlying multigraph of) G. A maximal vertex set $C \subseteq V(G)$ such that $G[C]$ is connected is called a *connected component* of G. We sometimes abuse notation and identify connected components C with their vertex-induced subgraphs $G[C]$. When it is clear from the context we denote connected components simply by *components*. By the *component graph* $\mathbb{C}_G$ of G we denote the complete graph that has a vertex for every connected component of G. Consider a trail t that traverses only vertices of G and the trail s in $\mathbb{C}_G$ that is obtained from t as follows: for every connected component C of G, substitute every maximum length subtrail t' of t such that $V(t') \subseteq C$ by the vertex in $\mathbb{C}_G$ that corresponds to C. We denote the underlying trail of s by $\mathbb{C}_G(t)$. For an example on connected components and the $\mathbb{C}_G()$ mapping, see Figure 1.1.

Degree and Balance. Let $G = (V, A)$ be a directed multigraph. The *indegree* (*outdegree*) of a vertex $v \in V$ denoted by $\text{indeg}(v)$ ($\text{outdeg}(v)$) is $|\{(u, v) \in A\}|$ ($|\{(v, u) \in A\}|$). The *balance* of v, denoted by $\text{balance}(v)$, is $\text{indeg}(v) - \text{outdeg}(v)$. In directed multigraphs, a vertex v is called *balanced* if $\text{balance}(v) = 0$.

Let $G = (V, E)$ be a multigraph. The *degree* of a vertex $v \in V$ denoted by $\text{deg}(v)$ is $|\{\{u, v\} \in E\}|$. In multigraphs, a vertex v is called *balanced* if $\text{deg}(v)$ is even.

Let $G = (V, E)$ be a (directed) multigraph. We denote the set of unbalanced vertices by I_G. If G is directed, we denote $\{v \in V : \text{balance}(v) > 0\}$ by I_G^+ and $\{v \in V : \text{balance}(v) < 0\}$ by I_G^-.

Eulerian Graphs and Extensions. A closed trial in a (directed) multigraph G is said to be *Eulerian*, if it traverses every edge in $E(G)$ (arc in $A(G)$) exactly once and every vertex in $V(G)$ at least once.[2] A (directed) multigraph is called Eulerian, if it contains an Eulerian trail. The following theorem holds.

Theorem 1.1.1. *A (directed) multigraph is Eulerian if and only if it is connected and every vertex is balanced.*

A version of Theorem 1.1.1 that is restricted to graphs is due to Euler, a proof for the generalized version above can be found in Bang-Jensen and Gutin [2]. We call an edge multiset (arc multiset) E such that $G + E$ is Eulerian an *Eulerian extension* for G. Edges (arcs) contained in E are called *extension edges* (*extension arcs*).

Vertex Partitions and Bipartite Graphs. Let $G = (V, A)$ be a (directed) multigraph. A family of sets $P = \{C_1, \ldots, C_k\}$ is called a *vertex partition* of G, if $V = \bigcup_{i=1}^{k} C_i$ and $C_i \cap C_j = \varnothing$ for all $1 \leq i < j \leq k$. The sets C_i, $1 \leq i \leq k$ are called *cells* of the partition P.

A graph $G = (V_1 \uplus V_2, E)$ is called a *bipartite graph*, if $\{V_1, V_2\}$ is a vertex partition of G and for every $e = \{u, v\} \in E : u \in V_1 \wedge v \in V_2$.[3]

Matchings. Let $G = (V, E)$ be a graph. A set $M \subseteq E$ is called a *matching* in G or of the vertices in G, if for every $e, f \in E : e \cap f = \varnothing$. A matching M is called *perfect* if for every vertex $v \in V$ there is an edge in M that is incident to v. The following theorem holds:

Theorem 1.1.2 (Hall's condition)**.** *A bipartite graph $G = (V_1 \uplus V_2, M)$ has a perfect matching, if and only if $U \leq N(U)$ for every $U \subseteq V_1$. Here, $N(U)$ denotes the set of all neighbors of U.*

A proof for Theorem 1.1.2 can be found in Bang-Jensen and Gutin [2].

---

[2]Note that there seem to be two equally well-accepted definitions of Eulerian trails: The definitions with and without the additional vertex condition. We chose the one with the vertex condition here, because it makes it easier to deal with connected components that consist only of one vertex. Algorithmically, problems according to both formulations are easily inter-transformable.

[3]In this regard we use the symbol $\uplus$ to indicate a disjoint union of the vertex sets.

Eulerian extension on unweighted graphs		
	Connected	Disconnected
Undirected	$\overline{m}\sqrt{n}$	$\overline{m}\sqrt{n}$
Directed	$n\overline{m}\log(n)$	$\overline{m}\log(n)(\overline{m} + n\log(n))$

Table 1.1: Complexity results regarding unweighted Eulerian extension problems. The number of edges in complement graphs of graphs with m edges is denoted by $\overline{m}$. Running times in big-O notation. The result for undirected and directed graphs have been obtained by Boesch et al. [4] and Dorn et al. [10], respectively.

1.2 Problems, Variants, Relationships

Eulerian graphs are interesting by themselves from a graph-theoretic point of view. However, they also bear intuitive and practical applications. In this section we introduce various problems regarding Eulerian graphs, their complexity if it is known, and point out relations to other problems.

As we will see later in this section, some natural problems translate into the problem of making a given graph Eulerian by adding edges or arcs, respectively. In these problems it is beneficial to add as few edges as possible, or to add edges such that their total weight is as low as possible. This translates into the following problem formulation:

> EULERIAN EXTENSION (EE)
> *Input:* A directed multigraph $G = (V, E)$ and a weight function $\omega :$
> $\quad V \times V \to [0, \omega_{max}] \cup \{\infty\}$.
> *Question:* Is there an Eulerian extension for G of weight at most ω_{max}?

The problem of UNWEIGHTED EULERIAN EXTENSION is EE where every arc in $V \times V$ has weight 1. Natural variants of these problems can be derived by substituting undirected multigraphs, directed graphs or graphs for multigraphs in the problem description. As we will also see, the complexity of problems regarding weighted Eulerian extensions depends heavily on the connectedness of the input graph. So, connectedness makes for another intuitive distinction in these problems.

Polynomial-time Solvable Variants. Table 1.1 shows polynomial running time results for unweighted Eulerian extension problems on graphs. For unweighted multigraphs, Dorn et al. [10] obtained linear-time algorithms for both the directed and undirected case. These algorithms work regardless of whether the input multigraph is connected or disconnected. Polynomial-time solvability has also been proven for the unweighted and connected variants shown in Table 1.2.

Fixed-Parameter Tractability. In general, EE is NP-hard. We recapitulate two NP-hardness proofs in the following subsections. However, Dorn et al. [10] have proven EE to be fixed-parameter tractable with respect to a slightly complicated parameterization: Let $\mathbb{E}(G, \omega)$ be the set of all Eulerian extensions E for the directed multigraph G with weight $\omega(E) \leq \omega_{max}$ according to the weight function ω.

| | Weighted, connected Eulerian extension | |
	Graphs	Multigraphs
Undirected	$|I_G|^3 \log(|I_G|)$	$|I_G|^3 \log(|I_G|)$
Directed	$\overline{m} \log(n)(\overline{m} + n \log(n))$	$n^3 \log(n)$

Table 1.2: Complexity results regarding weighted Eulerian extension problems on connected graphs. The number of edges in complement graphs of graphs with m edges is denoted by $\overline{m}$. Recall that I_G denotes the set of not balanced vertices in the input graph. Running times in big-O notation. These results have been obtained by Dorn et al. [10].

Theorem 1.2.1. EULERIAN EXTENSION *parameterized by* $k = \max\{|E| : E \in \mathbb{E}(G, \omega)\}$ *is solvable in* $\mathrm{O}(4^k n^4)$ *time.*

Note that the according parameterization is likely not polynomial-time computable. This calls for the trick to encode the parameter in the corresponding language Q of the parameterized problem. The parameter then has to be checked for correctness by any algorithm that decides Q.

1.2.1 Relations to the Rural Postman Problem

In this section, we briefly review the many-one reductions from EULERIAN EXTENSION (EE) to the RURAL POSTMAN problem and back, given by Dorn et al. [10]. From these reductions we get parameterized equivalence with respect to parameters that motivate our choice of parameters for EE. The RURAL POSTMAN problem is defined as follows.

> RURAL POSTMAN (RP)
> *Input:* A directed graph $G = (V, A)$, a set $R \subseteq A$ of required arcs and a weight function $\omega : A \to [0, \omega_{max}] \cup \{\infty\}$.
> *Question:* Is there a walk W in G such that W traverses all arcs in R and $\omega(W) \leq \omega_{max}$?

Dorn et al. [10] observed that RP parameterized by the "number of arcs in the sought walk" and EE parameterized by "number of arcs in the sought Eulerian extension" are equivalent.[4] We take a brief look at their construction here and observe a further parameterized equivalence. The main idea in both reductions is to exploit the following observation.

Observation 1.2.1. *Let G be a directed graph and let W be a multiset of arcs in G. There is a closed walk in G that uses exactly the arcs W if and only if the directed multigraph $(V(G), W)$ is Eulerian.*

With Observation 1.2.1 it is easy to see that the following two constructions are polynomial-time many-one reductions from RP to EE and from EE to RP, respectively.

Construction 1.2.1. Let the directed graph $G = (V, A)$, the required arc set R and the weight function $\omega : V \times V \to [0, \omega_{max}] \cup \{\infty\}$ constitute an instance of RP.

---

[4]The actual parameters are slightly more complicated, but this intuition suffices here.

Construct an instance of EE by defining the directed multigraph $G' := (V, R)$ and a weight function $\omega' : V \times V \to [0, \omega_{max} - \omega(R)] \cup \{\infty\}$ by

$$\omega' := \begin{cases} \omega(a), & a \in A \land \omega(a) \leq \omega_{max} - \omega(R), \\ \infty, & \text{otherwise.} \end{cases}$$

Construction 1.2.2. Let the directed multigraph $G = (V, A)$ and the weight function $\omega : V \times V \to [0, \omega_{max}] \cup \{\infty\}$ constitute an instance of EE. Construct an instance of RP by defining the directed graph $G := (V, V \times V)$, the required arc set $R := A$, the weight function $\omega' = \omega$ and the maximum weight $\omega'_{max} := \omega_{max} + \omega(R)$.

In the search for suitable parameters for EE, we observed the following. Intuitively, we expect the number of connected components in $G\langle R \rangle$ to be small in practical instances. For instance, consider a postman's tour in a city that comprises a number of suburbs. The number of streets that have to be serviced in each of the suburbs is expected to be much higher than the streets in-between, thus forming connected components in each suburb. We also expect the sum of positive balances of all vertices in $G\langle R \rangle$ to be small: This sum is at most proportional to the number of required arcs, and we assume this number to be small compared to n in practice. With regard to EE, the following observation is of much interest.

Observation 1.2.2. *Let G be the input digraph and R the required arcs in an instance of* RP. *Let G' be the input graph in an instance of* EE. *Construction 1.2.1 and Construction 1.2.2 are polynomial-time polynomial-parameter many-one reductions with respect to the parameters*
(i) number of connected components in $G\langle R \rangle$ and number of connected components in G', and/or
(ii) sum of all positive balances in $G\langle R \rangle$ and sum of all positive balances in G'.

This motivates the analysis of EE with respect to these two parameters. In this regard, Frederickson [18] has proven the following theorem.

Theorem 1.2.2. RURAL POSTMAN *can be solved in $\mathrm{O}(n^3 n^{2c-2}/c!)$ time, where c is the number of connected components in $G\langle R \rangle$—the graph G being the input graph and R the set of required arcs.*

From this theorem it immediately follows that RP parameterized by the number of components in $G\langle R \rangle$ is in XP and thus, by Observation 1.2.2, EE parameterized by the number of components in the input graph also is in XP.

1.2.2 Relations to the Hamiltonian Cycle Problem

In this section we observe that the difficulty of solving EULERIAN EXTENSION (EE) depends on the number of components in the input graph. This is done using a reduction from the HAMILTONIAN CYCLE problem. A natural question is, whether the difficulty of solving EE depends *only* on the number c of components, that is, whether EE is fixed-parameter tractable with respect to the parameter c. We attack this question in Chapter 2, especially Section 2.3.

This section also shows that the parameter "sum of all positive balances of vertices in the input graph" for EE will likely not yield fixed-parameter tractability.

The HAMILTONIAN CYCLE problem is defined as follows.

Definition 1.2.1. Let G be a directed graph. A cycle in G is called *Hamiltonian* if it traverses every vertex in G exactly once.

> HAMILTONIAN CYCLE (HC)
> *Input:* A directed graph G.
> *Question:* Is there a Hamiltonian cycle in G?

Orloff [28] notes that the complexity of RP seems to depend on the number of connected components in $G\langle R\rangle$, where G is the input graph and R is the set of required arcs. In a way, Lenstra and Kan [24] proved this statement by giving a reduction from the NP-hard [23] HC problem such that the number of components in $G\langle R\rangle$ in the RP instance is exactly the number of vertices in the HC instance. In this section, we give a reduction from HC to EE illustrating that the same is true for EE.

The main idea of the reduction is that any Eulerian extension for EE has to connect all connected components in the input graph. Thus, we model every vertex by a connected component consisting of two vertices that are connected by two arcs: One arc in either direction. To model edges in the instance of RP, we utilize the weight function and choose ω_{max} accordingly to ensure that every feasible Eulerian extension is a cycle.

Construction 1.2.3. Let the directed graph $G' = (V', A')$ constitute an instance of RP. Construct an instance of EE as follows:

Define the directed multigraph $G = (V, A)$ by $V := V' \times \{0, 1\}$ and

$$A := \{((v, 1), (v, 0)), ((v, 0), (v, 1)) : v \in V'\}.$$

Set the maximum weight $\omega_{max} := |V'|$ and define the weight function ω by

$$\omega(a) := \begin{cases} 1, & a = ((u, 0), (v, 0)) \wedge (u, v) \in A', \\ \infty, & \text{otherwise.} \end{cases}$$

It is easy to see that this construction is correct using the following observation:

Observation 1.2.3. *Any Eulerian extension E for G with $\omega(E) \leq \omega_{max}$ is a cycle.*

Proof. Since E has to connect $|V'|$ connected components in G, it contains at least $|V'| - 1$ arcs. The Eulerian extension E cannot contain a maximum-length trail that is open, since there are no unbalanced vertices in G. For sake of contradiction assume that E contains three arcs that are incident to one vertex v in G. Then, to connect the remaining connected components in G via a closed trail, E has to contain at least $|V'| - 3$ arcs. However, then v is still not balanced and E has to contain at least one additional arc, totalling in $|V'| + 1$ arcs. Thus, by contradiction, every vertex in G has at most two incident arcs in E and thus E is a cycle. □

Thus, Construction 1.2.3 is correct and we have that the difficulty in solving EE depends on the number of components in the input graph. But the reduction given by Construction 1.2.3 also gives the following observation.

Observation 1.2.4. EE *parameterized by the sum of all positive balances of vertices in the input graph is not contained in XP, unless $P = NP$.*

Proof. Observe that all vertices in the graph G produced by Construction 1.2.3 are balanced. If EE parameterized by the sum b of all positive balances of vertices in the input graph was in XP, in particular all instances with $b = 0$ were solvable within polynomial time. Thus, HC would be solvable within polynomial time. □

1.2.3 Constrained Eulerian Extensions

A natural modification of Eulerian extension problems is to give constraints on the set of edges or arcs that can be added to the input graph in order to make it Eulerian. Note for example that in EULERIAN EXTENSION on graphs we can regard the condition that the input graph has to remain a graph with the added edges as a constraint on the allowed edges (that is multiedges are forbidden). Thusly constrained problems might also be interesting in practice. For instance, Höhn et al. [21] observed that the following class of constrained Eulerian extension problems has applications to sequencing problems:

> d-DIMENSIONAL EULERIAN EXTENSION
> *Input:* A directed graph $G = (V, A)$, where $V \subset \mathbb{Q}^d$.
> *Question:* Is there an Eulerian extension E for G such that for every $(u, v) \in E$ it holds that $u \geq v$ component-wise?

However, Höhn et al. [21] also have proven that d-DIMENSIONAL EULERIAN EXTENSION is NP-complete. We model such constraints on the extension edges in such problems as instances of EULERIAN EXTENSION by simply defining the weight function accordingly—assigning forbidden arcs or edges the weight ∞, and setting the maximum weight to a large enough value.

We use d-DIMENSIONAL EULERIAN EXTENSION as a helper problem in Chapter 3. In order to deal conveniently with the arc constraints, we introduce some notation at this point.

Definition 1.2.2. Let ω be a weight function assigning weights $[0, \omega_{max}] \cup \{\infty\}$ to arcs. An arc a is called *allowed* with respect to ω if $\omega(a) < \infty$. If the weight function is clear from the context, then we simply say that the arc is allowed.

1.3 Our Work

In recent research by Dorn et al. [10] the problem EULERIAN EXTENSION (EE) has been shown to be fixed-parameter tractable with respect to the parameter $k =$ "number of arcs in the sought Eulerian extension".[5] In this work we initiate a more fine-grained analysis of the EE problem by considering parameters that are upper bounded by k. In particular, we study the parameterizations "number c

---

[5]The actual parameter is slightly more complicated—see page 11—but the intuition of the number of needed extension arcs suffices here.

Parameterized complexity results for EULERIAN EXTENSION		
Parameter	Known	New
k	$\in \mathrm{FPT} : 4^k$	no polykernel
c	$\in \mathrm{XP}$	$\in \mathrm{W}[P]$, no polykernel
b, c	—	$\in \mathrm{FPT} : 4^{c \log(bc^2)}$, no polykernel

Table 1.3: Overview on parameterized complexity results for EE regarding various parameters. Fixed-parameter tractability results include the superpolynomial term of the corresponding algorithm. Known results: The fixed-parameter tractability result for parameter k is due Dorn et al. [10]. The XP-result for parameter c is due Frederickson [18] (see Subsection 1.2.1). New results: The fixed-parameter tractability result for the combined parameter b, c is shown in Theorem 2.2.3 and Corollary 2.2.1. The W[P]-result for parameter c follows from Observation 2.3.1 and Theorem 2.3.5. The non-existence of polynomial-size problem kernels is shown in Theorem 3.3.2 and its corollaries.

of components in the input graph" and "sum b of all positive balances of vertices in the input graph". Since any Eulerian extension E for a multigraph has to produce a connected graph, it holds that $|E| \geq c - 1$ and thus $k \geq c - 1$. Also, any Eulerian extension E has to balance all vertices in the given multigraph, that is, for every vertex v with balance $d > 0$, it has to contain at least d outgoing arcs. Hence it holds that $|E| \geq b$ and thus $k \geq b$. Table 1.3 gives a compact overview over the new and known results regarding EE.

EE parameterized only with b is already NP-hard when $b = 0$: Consider the reduction we give in Subsection 1.2.2 to prove NP-hardness for EE. This reduction produces instances with $b = 0$. Also, the question whether EE is fixed-parameter tractable when parameterized by c is a long-standing open question which arose implicitly in research by Frederickson [18, 19]. His work implies that EE is polynomial-time solvable for every constant value of c (see Subsection 1.2.1). However, his algorithm does not imply fixed-parameter tractability and this question seems to be hard to answer. Nevertheless, in Chapter 2 we show that when parameterizing with both b and c EE becomes fixed-parameter tractable.

Pursuing the question whether EE is fixed-parameter tractable with only the parameter c, we restate EE in the context of matchings in Chapter 2 and show that the problem CONJOINING BIPARTITE MATCHING is parameterized equivalent to EE. Using the matching formulation we obtain a fixed-parameter tractability result for a restricted class of EE when parameterized by c.

We also consider preprocessing routines for EE in Chapter 3. In this regard, we show that d-DIMENSIONAL EULERIAN EXTENSION does not admit a polynomial problem kernel with respect to the parameter k. The result also transfers to the parameters b, c and the more general problem EE.

Chapter 2

Connected Components

The main results given in this chapter are an efficient algorithm for EULERIAN EXTENSION (EE) with running time in $O(4^{c \log(bc^2)} n^2 (b^2 + n \log(n)) + n^2 m)$ and the parameterized equivalence of EE parameterized by c and CONJOINING BIPARTITE MATCHING. Here, c is the number of components and b is the sum of all positive balances in the input graph, that is for the input graph G, it is $b = \sum_{v \in I_G^+} \text{balance}(v)$. The equivalence to the matching problem also yields an algorithm for a restricted form of EE with $O(2^{c(c + \log(2c^4))} (n^4 + m))$ running time. The latter result represents some partial progress to answer the question of whether EE is fixed-parameter tractable with respect to the parameter c.

We first make some observations about Eulerian extensions in Section 2.1 which expose that every Eulerian extension corresponds to a specific structure that has an intimate relationship to the connected components of the input graph. This then leads to a modified problem derived from EE in Section 2.2. There we consider the problems EULERIAN EXTENSION WITH ADVICE (EEA) and EULERIAN EXTENSION WITH MINIMAL CONNECTING ADVICE (EECA) where the structure of the sought Eulerian extensions is made explicit in the input. These restricted problems seem to be easier to tackle and we derive an algorithm with $O(4^{c \log(b)} n^2 (b^2 + n \log(n)) + n^2 m)$ running time for EECA. Using observations about the relationship between EE and EECA we derive an algorithm for EE running in $O(4^{c \log(bc^2)} n^2 (b^2 + n \log(n)) + n^2 m)$ time.

In Section 2.3 we introduce CONJOINING BIPARTITE MATCHING (CBM) and show that it is tractable on some restricted graph classes. We give parameterized reductions from EE to CBM and from CBM to EE using some intermediary problems that we introduce in Section 2.2. This then yields the parameterized equivalence of CBM and EE. As simple corollaries, we derive fixed-parameter tractability of EE with respect to parameter c on some restricted input instances. The reductions also yield some results for intermediary problems, for example a problem kernel for EECA that has size polynomial in b and c.

Consult Figure 2.1 and Table 2.1 for an overview on the reductions given in this chapter and the tractability results obtained.
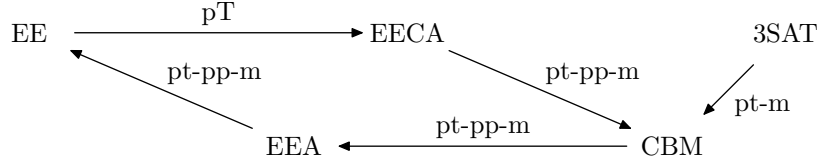
EE   → pT →   EECA   3SAT

pt-pp-m   pt-pp-m   pt-m

EEA   ← pt-pp-m ←   CBM

Figure 2.1: Schematic overview on the reductions given in this chapter. The label "pT" indicates a parameterized Turing reduction, the label "pt-pp-m" indicates a polynomial time polynomial parameter many-one reduction, and the label "pt-m" indicates a classical polynomial time many-one reduction. The reductions from and to EE are covered in Section 2.2. The reductions from and to CBM are given in Section 2.3.

| | Tractability results | |
Problem	Result	Proposition
CBM[a]	$n + m$	Corollary 2.3.1
CBM[b]	$2^{j(j+1)}n + n^3$	Lemma 2.3.4
EECA	$4^{c\log(b)}n^2(b^2 + n\log(n)) + n^2m$	Theorem 2.2.3
EEA	b^2c vertex kernel	Corollary 2.3.6
EE[c]	$16^{c\log(c)}(cn^4 + m)$	Corollary 2.3.3
EE[d]	$2^{c(c+\log(2c^4))}(n^4 + m)$	Corollary 2.3.4
EE	$4^{c\log(bc^2)}n^2(b^2 + n\log(n)) + n^2m$	Corollary 2.2.1

Table 2.1: Overview on tractability results given in this chapter. All values in big-O notation. Here, j denotes the parameter "join set size" in CBM instances. This parameter corresponds to the parameter "number of components" in EE instances in reductions we give in this chapter.

[a]When the input graph is a forest.
[b]When the bipartite input graph has maximum degree two in one of its cells.
[c]When the allowed arcs "resemble" a forest.
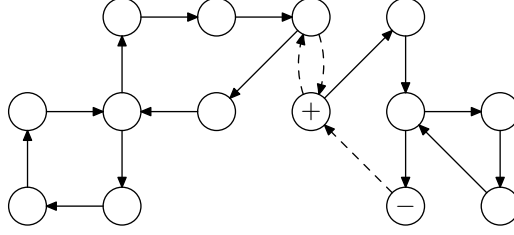[d]When the allowed arcs "resemble" a vertex-disjoint union of cycles.

Figure 2.2: Examples of a closed maximum length trail (left, solid arcs) and an open maximum length trail (right, solid arcs) in an Eulerian extension (solid arcs). Arcs belonging to the input graph are dashed. Observe that the vertex $+$ is the only vertex in I_G^+ and the vertex $-$ is the only vertex in I_G^-.

2.1 Structure of Eulerian Extensions

In this section, we show that we can assemble a minimum-weight Eulerian extension for a graph G using trails that are of restricted structure, and bound the length and number of "long" trails by polynomials in the number of components in G. To this end, we consider trails in Eulerian extensions.

We investigate preprocessing routines for instances of EE—namely, we split vertices (Transformation 2.1.1) and use shortest-path preprocessing (Transformation 2.1.2)—that allow us to modify any valid Eulerian extensions such that we can make assumptions about their trails without increasing the weight of the extensions. In this section, we frequently use the component graph $\mathbb{C}_G$ of a graph G and the mapping $\mathbb{C}_G(t)$ of trails t in G to trails in $\mathbb{C}_G$. These are defined on page 9 in Section 1.1. The main result of this section is as follows.

Theorem 2.1.1. *Let G be a directed multigraph with c connected components. Let G and the weight function $\omega : V \times V \to [0, \omega_{max}] \cup \{\infty\}$ constitute an instance of* Eulerian Extension *that is preprocessed using Transformation 2.1.1 and Transformation 2.1.2. Then, there is a set $S := \{t_1, \ldots, t_k\}$ of pairwise edge-disjoint paths and cycles each in the graph $(V, V \times V)$ such that*

(i) $\bigcup_{i=1}^{k} A(t_i)$ is an Eulerian extension of minimum weight for G,

(ii) each $t_i \in S$ contains at most $c + 1$ vertices,

(iii) in S there are at most $c(c-1)/4$ paths and cycles containing more than one arc,

(iv) the number of paths in S is at most $|I_G^+| = |I_G^-|$,

(v) for $t_i \neq t_j \in S$ of length at least two $\mathbb{C}_G(t_i)$, and $\mathbb{C}_G(t_j)$ are edge-disjoint,

(vi) the graph defined by the union of all trails $\mathbb{C}_G(t_1), \ldots, \mathbb{C}_G(t_n)$ without their initial vertices does not contain a cycle.

In this section, let $G = (V, A)$ be a directed multigraph, let E be an Eulerian extension for G—that is $G + E := (V, A \cup E)$ is Eulerian—and let the function $\omega : V \times V \to [0, \omega_{max}] \cup \{\infty\}$ be a weight function.

Observation 2.1.1. *A maximum-length trail in an Eulerian extension for a graph G either is closed or starts in I_G^+ and ends in I_G^-.*

Proof. Consider the initial vertex v_A and terminal vertex v_Ω of a trail t in the Eulerian extension E. The vertices v_A and v_Ω are balanced in $G + E$.

19

Assume that v_Ω is not balanced in G. Every time t traverses v_Ω, it uses one arc in E that enters v_Ω and one that leaves it. This implies that $v_\Omega \neq v_A$ because v_Ω is balanced in $G + E$ and thus there is an odd number of arcs in E incident to v_Ω (recall that t is of maximum length). Since t ends in v_Ω, this also implies that $v_\Omega \in I_G^-$. Analogously we get that $v_A \in I_G^+$.

Now assume that v_Ω is balanced in G. Since t cannot be extended, it already uses every arc incident to v_A and v_Ω. However, if v_Ω is not equal to v_A, there are more arcs entering v_Ω than leaving v_Ω in E. This means that v_Ω is not balanced in $G + E$ which is a contradiction. □

Figure 2.2 illustrates Observation 2.1.1.

Preprocessing Routines. There is a preprocessing routine introduced by Dorn et al. [10] that ensures that every vertex has balance between -1 and 1. This later helps to give a bound on very short trails in Eulerian extensions.

Transformation 2.1.1 (Splitting Vertices). Let the graph ($G = (V, A)$, the weight function ω and the maximum weight ω_{max}) constitute an instance of EE. Compute a new instance as follows: Search for a vertex v with $|\,\text{balance}(v)| > 1$, introduce a new vertex u. If $\text{balance}(v) > 0$, choose an arbitrary arc (w, v), delete it and add the arc (w, u). Proceed analogously, if $\text{balance}(v) < 0$. Add the arcs $(u, v), (v, u)$. Finally, define a new weight function ω' for each pair of vertices $x, y \in V$ as follows.

$$\omega'(x, y) = \begin{cases} \infty, & x = u, y = v \lor x = v, y = u \\ \omega(v, y), & x = u \\ \omega(x, v), & y = u \\ \omega(x, y), & \text{otherwise} \end{cases}$$

Lemma 2.1.1. *Transformation 2.1.1 is correct, that is, it maps yes-instances and only yes-instances to yes-instances. Also, Transformation 2.1.1 can be applied exhaustively in $O(n^2 m)$ time. When applied exhaustively, the resulting instance contains only vertices v with $|\,\text{balance}(v)| \leq 1$.*

Proof. The last statement of the lemma is clear. Concerning the running time, we can iterate over every vertex $v \in V$ ($O(n)$ time), check if it has high absolute balance ($O(m)$ time) and, if so, perform the weight function update ($O(n)$ time) and perform the local modifications ($O(1)$ time) for every "excess arc" incident to v (there are at most m many). In total, this is $O(n^2 m)$ time.

To prove the correctness, we only have to examine one application of Transformation 2.1.1: Let ($G' = (V', A'), \omega', \omega_{max}$) be an instance of EE where Transformation 2.1.1 has been applied once at vertex v yielding the new vertex u. Given an Eulerian extension for the input graph G, we can obtain an Eulerian extension for G' of the same weight by modifying an arc $a \in E$ incident to v appropriately such that it starts or ends in u. If we are given an Eulerian extension for G', at least one arc in it has to be incident to u and thus we can obtain an Eulerian extension for G by modifying it to start or end in v. □

We can apply a further preprocessing routine to make some further observations about trails in Eulerian extensions:

Transformation 2.1.2 (Shortest-Path Preprocessing)**.** For an input instance of EE consisting of the graph $G = (V, A)$, the weight function ω and the maximum weight ω_{max}, derive a new instance by computing a new weight function ω' as follows:

$\omega'(u, v) :=$ weight of a shortest path from u to v in the graph $(V, V \times V)$.

Lemma 2.1.2. *Transformation 2.1.2 is correct—that is, it maps yes-instances and only yes-instances to yes-instances—and can be applied in* $O(n^3)$ *time.*

Proof. It is clear that for any Eulerian extension E of G it holds that $\omega'(E) \leq \omega(E)$, making any feasible Eulerian extension in the original instance also one for the modified instance. Now let E be an Eulerian extension for G with $\omega'(E) \leq \omega_{max}$. We get an Eulerian extension E' for G with $\omega(E') \leq \omega_{max}$ by exchanging every arc $a = (u, v) \in E$ with $\omega'(a) < \omega(a)$ by the set of arcs of a shortest path from u to v in the graph $(V, V \times V)$ with respect to the weight function ω.

Using Dijkstra's algorithm we can compute in $O(n^2)$ time the weights of the shortest paths between one vertex v and any other in G and update the weight function accordingly. Doing this for every vertex in G takes $O(n^3)$ time. $\square$

Shortest-path preprocessing and splitting vertices enables us to make a range of useful observations regarding trails in Eulerian extensions. In the following we assume any instance of EULERIAN EXTENSION to be preprocessed using Transformation 2.1.1 and Transformation 2.1.2. In the subsequent sections, we use this preprocessing in parameterized algorithms and reductions. Thus, we need to know whether it is parameter-preserving. This is the case, as the following observation shows.

Observation 2.1.2. *The number of components and the sum of all positive balances of vertices in an instance of* EE *are invariant under Transformation 2.1.1 and Transformation 2.1.2.*

Shortcutting Trails in Eulerian Extensions. Using Transformation 2.1.2, we can define the following transformation that operates on trails of an Eulerian extension.

Transformation 2.1.3. Let E be an Eulerian extension of G, let t be a trail in the graph $(V(G), E)$ and let s be a subtrail of t where s has the initial vertex v_A and the terminal vertex v_Ω. Obtain a new trail t' by substituting the edge (v_A, v_Ω) for s in t and derive a new arc set E' by substituting $A(t')$ for $A(t)$ in E. Define $\text{shortcut}(E, t, s) := (E', t')$.

Figure 2.3 illustrates Transformation 2.1.3.

Lemma 2.1.3. *Let* $\text{shortcut}(E, t, s) = (E', t')$ *where the trail s has initial vertex v_A and terminal vertex v_Ω. The following statements hold:*
  (i) $\omega(E') \leq \omega(E)$.
  (ii) Every vertex in $V(s)$ is balanced in $G + E'$.
  (iii) If every vertex of s except v_A and v_Ω is contained in a connected component of G that also contains a vertex of t', then the arc set E' is an Eulerian extension for G.
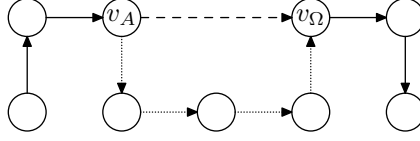
21

Figure 2.3: Example of an application of Transformation 2.1.3. Solid arcs and dotted arcs belong to t, dotted arcs to s and the dashed arc is substituted for the dotted arcs in t'.

Proof. Statement (i) is trivial because of the implicitly transformed weight function (Transformation 2.1.2).

By substituting (v_A, v_Ω) for s, every vertex on s except v_A and v_Ω looses one indegree and one outdegree. Hence, augmenting G with E' results in a graph without unbalanced vertices (statement (ii)).

For statement (iii) it remains to show that the graph $(V(G), A \cup E')$ is connected: If every vertex of s except v_A and v_Ω is contained in a connected component of G that also contains another vertex of t', then augmenting G with E' results in a connected graph, making E' an Eulerian extension for G (Theorem 1.1.1). □

Observation 2.1.3. *For any Eulerian extension E of $G = (V, A)$ there is an Eulerian extension E' of at most the same weight such that any trail with arcs in E' visits every vertex at most once.*

Proof. Assume that in the Eulerian extension E there is a trail t that visits $v \in V$ more than once. Then there is a subtrail

$$s = (u, (u, v), v, (v, w), w)$$

of t with $u, w \in V$. Let $(\hat{E}, t') = \text{shortcut}(E, t, s)$. By Lemma 2.1.3, $\hat{E}$ is an Eulerian extension for G because t' still visits v (one time less than t). If we recursively shortcut edges in trails in E until every such trail visits any vertex at most once, we obtain an Eulerian extension E'. By Lemma 2.1.3, $\omega(E') \le \omega(E)$. □

Observation 2.1.3 allows us to assume trials in Eulerian extensions to be cycles when they are closed and paths otherwise.

Observation 2.1.4. *For any Eulerian extension E of G, there is an Eulerian extension E' of at most the same weight such that for any path p and any cycle c in E' such that p and c are edge-disjoint and have length at least two the following statements hold:*

(i) p and c do not successively visit two vertices contained in exactly one connected component of G.

(ii) p and c do not visit one connected component of G twice except for the initial and terminal vertex.

(iii) p and c have length at most the number of connected components of G.

Proof. The proof for (i) and (ii) is similar to the proof of the observation above. Again we can shortcut edges and obtain an Eulerian extension of at most the same weight. Statement (iii) directly follows from (i) and (ii). □
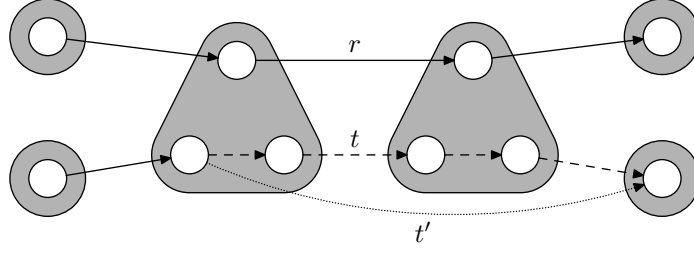
Figure 2.4: Gray objects represent components of G. Shown are two trails r (top) and t (bottom, solid and dashed arcs) in an Eulerian extension. The trails $\mathbb{C}_G(r), \mathbb{C}_G(t)$ share two vertices. The dashed arcs represent a subtrail s' of t as in Lemma 2.1.4 and thus we can obtain a path t' (bottom, solid and dotted arcs) replacing t, while maintaining connectedness and balance of all vertices.

Shortcutting and Component Graphs. We can further extend our observations by looking at component graphs $\mathbb{C}_G$ and the mapping of trails t in G to trails $\mathbb{C}_G(t)$ in $\mathbb{C}_G$. Recall these definitions stated on page 9 in Section 1.1. The following lemma is a generalization of statement (iii) in Lemma 2.1.3.

Lemma 2.1.4. *Let E be an Eulerian extension of G, let t, r be trails in the directed multigraph $(V(G), E)$ such that the trails $\mathbb{C}_G(r)$ and $\mathbb{C}_G(t)$ are not vertex-disjoint. Furthermore, let s be a subtrail of t in the directed multigraph $(V(G), E)$ such that $\mathbb{C}_G(s)$ is a subtrail of $\mathbb{C}_G(r)$. Let s' be a subtrail of t such that s is a subtrail of s' and s traverses exactly one vertex less than s'. Set $(E', t') = $ shortcut(E, t, s'). Then E' is an Eulerian extension for G.*

Proof. Lemma 2.1.3 shows that the vertices in $G + E'$ are balanced. It remains to show that the resulting graph is connected: Any connected component that is traversed by s is also traversed by u. The trails $\mathbb{C}_G(u)$ and $\mathbb{C}_G(t')$ still share a vertex and thus $G + E'$ is connected. $\qquad\square$

Lemma 2.1.4 leads to the following Observation 2.1.5, which is illustrated in Figure 2.4.

Observation 2.1.5. *For any Eulerian extension E of G there is an Eulerian extension E' of at most the same weight such that for any two edge-disjoint trails t_1, t_2 in E' it holds that $\mathbb{C}_G(t_1), \mathbb{C}_G(t_2)$ either are vertex-disjoint, share at most one vertex, or share only their initial and terminal vertices.*

Proof. This follows directly from Lemma 2.1.4 by shortcutting subtrails that are shared by two such trails in $\mathbb{C}_G$. $\qquad\square$

We can improve this even to the following.

Observation 2.1.6. *For any Eulerian extension E of G there is an Eulerian extension E' of at most the same weight such that for any set of edge-disjoint trails $\{t_1, \ldots, t_n\}$ in E' it holds that the edge-induced graph $\mathbb{C}_G\langle \bigcup_{i=1}^{n} \mathbb{C}_G(t_i)'\rangle$ does not contain a cycle as subgraph, where $\mathbb{C}_G(t_i)'$ is $\mathbb{C}_G(t_i)$ without the initial vertex.*

Proof. By Observation 2.1.3 we may assume that $S := \{t_1, \ldots, t_n\}$ are paths or cycles. Assume that $\mathbb{C}_G(\bigcup_{i=1}^n \mathbb{C}_G(t_i)')$ contains a cycle c and that S is minimal with respect to this property. Let $e \in t_i$ be an arbitrary edge on c. There is a subtrail s of t_i such that $\mathbb{C}_G(s)$ traverses e and at least one edge not belonging to c—recall that $\mathbb{C}_G(t_i)'$ is $\mathbb{C}_G(t_i)$ without the initial vertex. Shortcutting s maintains balance of every vertex (statement (ii), Lemma 2.1.3) and connectedness, because afterwards $\mathbb{C}_G(t_i)$ is not vertex-disjoint from c. Since an edge is removed from c, it is a path after shortcutting s.

Iterating the shortcutting for every cycle in the graph $\mathbb{C}_G(\bigcup_{i=1}^n \mathbb{C}_G(t_i)')$ eventually removes every cycle after a finite amount of steps, because obviously the statement of the lemma holds true, if $t_1, \ldots, t_n$ have length one, and because in every step the number of arcs in E decreases by at least one. $\square$

We use Observation 2.1.6 in forthcoming Subsection 2.2.2 to efficiently derive the structure of a suitable Eulerian extension for a given graph. We are now ready to prove Theorem 2.1.1.

Theorem 2.1.1. Let G be a directed multigraph with c connected components. Let G and the weight function $\omega : V \times V \to [0, \omega_{max}] \cup \{\infty\}$ constitute an instance of EULERIAN EXTENSION that is preprocessed using Transformation 2.1.1 and Transformation 2.1.2. Then, there is a set $S := \{t_1, \ldots, t_k\}$ of pairwise edge-disjoint paths and cycles each in the graph $(V, V \times V)$ such that

(i) $\bigcup_{i=1}^k A(t_i)$ is an Eulerian extension of minimum weight for G,

(ii) each $t_i \in S$ contains at most $c + 1$ vertices,

(iii) in S there are at most $c(c-1)/4$ paths and cycles containing more than one arc,

(iv) the number of paths in S is at most $|I_G^+| = |I_G^-|$,

(v) for $t_i \neq t_j \in S$ of length at least two $\mathbb{C}_G(t_i)$, and $\mathbb{C}_G(t_j)$ are edge-disjoint,

(vi) the graph defined by the union of all trails $\mathbb{C}_G(t_1), \ldots, \mathbb{C}_G(t_n)$ without their initial vertices does not contain a cycle.

Proof. We simply take an Eulerian extension E of minimum weight for the directed multigraph G and successively remove maximum-length paths from E to obtain a set of trails $S = \{t_1, \ldots, t_k\}$. The sought properties of the trails follow from the observations we made in this section: Statement (i) is trivial. From Observation 2.1.3 we can assume that each t_i either is a path or a cycle. The maximum-length $c + 1$ of maximum-length cycles and paths (statement (ii)) can be assumed because, by Observation 2.1.4, we can assume that each trail traverses at most one vertex in each component except the terminal vertex. Statement (v) follows directly from Observation 2.1.5. The maximum number of maximum-length paths p and cycles d of length at least two (statement (iii)) can be assumed because we can assume that $\mathbb{C}_G(p), \mathbb{C}_G(d)$ use two edges (Observation 2.1.4), they are edge-disjoint (Observation 2.1.5) and there are at most $c(c-1)/2$ edges in $\mathbb{C}_G$. The upper bound $|I_G^+|$ on the number of maximum-length paths (statement (iv)) can be assumed because every vertex v has $|\text{balance}(v)| \leq 1$ (Lemma 2.1.1) and each such path starts and ends in an unbalanced vertex (Observation 2.1.1). Finally, statement (vi) follows directly from Observation 2.1.6. $\square$

2.2 Simplification through Advice

In Section 2.1 we observed that any Eulerian extension can be modified to conform to a restricted structure with respect to the connected components in the input graph. We will observe in Chapter 3, that this structure cannot be determined within polynomial time—unless $\text{coNP} \subseteq \text{NP/poly}$, which seems unlikely. There, we implicitly use that fact, that it is not clear how components are connected through an Eulerian extension in order to obtain lower bounds for problem kernels. An obvious question is, whether the structure of an Eulerian extension can be determined using fixed-parameter algorithms whose super-polynomial-time portion depends only on the connected components of the input graph. This question is considered in the following sections.

We consider the general problem EULERIAN EXTENSION (EE), and investigate its connection to the problem EULERIAN EXTENSION WITH ADVICE (EEA) in which the structure of allowed Eulerian extensions may be given by the input. In order to get a grasp at the structure of Eulerian extensions, we introduce the notion of hints and advice:

Definition 2.2.1. Let $G = (V, A)$ be a directed multigraph. A *hint* for G is an undirected path or cycle t of length at least one in the component graph $\mathbb{C}_G$ together with the information that t shall form a cycle of a path in an Eulerian extension of G.[1] We call the corresponding hints *cycle hints* and *path hints*, respectively. We say a set of hints P is an *advice* to the graph G if the hints are edge-disjoint.[2] We say that a path p in the graph $(V, V \times V)$ *realizes* a path hint h if $\mathbb{C}_G(p) = h$ and the initial vertex of p has positive balance and the terminal vertex has negative balance in G. We say that a cycle c in the graph $(V, V \times V)$ realizes a cycle hint h if $\mathbb{C}_G(c) = h$. We say that an Eulerian extension E *heeds the advice* P if it contains paths and cycles that realize all hints in P.

Now consider the following restricted version of EE:

> EULERIAN EXTENSION WITH ADVICE (EEA)
> *Input:* A directed multigraph $G = (V, A)$ with a weight function $\omega :$
> $V \times V \to [0, \omega_{max}] \cup \{\infty\}$ and advice P.
> *Question:* Is there an Eulerian extension E of G that is of weight at
>   most ω_{max} and heeds the advice P?

For an example of an instance of EEA, see Figure 2.5. The EEA problem may be interesting in practical applications where the structure of a sought Eulerian extension is partly known. However, our intent is to use this problem to make the complete structure of the Eulerian extension explicit. We derive efficient algorithms that guess the structure as advice and then realize each hint.

In Subsection 2.2.1, we simplify EEA and gather a useful tool for its analysis. Then, in Subsection 2.2.2, we look at the relationship of EE and EEA. We introduce a variant of EEA that seems to be easier to tackle than EE. In Subsection 2.2.3, we give an efficient algorithm for this variant that also transfers over to EE.

---

[1] The extra information is necessary because a hint to a path may be a cycle in $\mathbb{C}_G$.

[2] Note that there is a difference between advice in our sense and the notion of advice in computational complexity theory. There a piece of advice applies to every instance of a specific length.
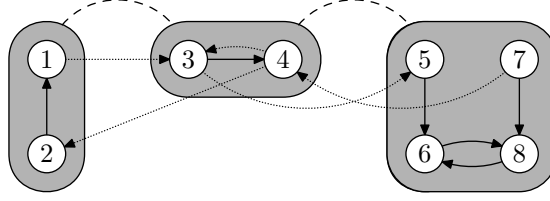
Figure 2.5: An instance of EEA comprising the vertices 1 through 8 and the solid arcs. Gray objects represent components of the input graph G and the the dashed lines are a hint h that forms a piece of advice $P = \{h\}$ for G. The dotted arcs form an Eulerian extension E of G. Both the paths traversing the vertices $1, 3, 5$ and $7, 4, 2$ realize h. Thus, E heeds P.

In the following sections, we assume all instances of EE and EEA to be preprocessed using Transformation 2.1.1 ("splitting vertices") and Transformation 2.1.2 ("shortest-path preprocessing") as introduced in Section 2.1. We give parameterized reductions that use the parameters number of components and sum of all positive balances of vertices in the input graph. For these one can assume without loss of generality that the instances are preprocessed using the two transformations, because of Observation 2.1.2.

2.2.1 Computing Realizations of Hints

In this subsection, we introduce the minpath function, which calculates minimum-weight paths that consist of allowed arcs and traverse connected components in a specific order. Using this function, we show that EEA and the problem Eulerian Extension with Cycle-free Advice (EE∅A) are equivalent under polynomial-time many-one reductions. That is, a minimum-weight realization for any hint to a cycle can be found in polynomial time. We use this equivalence in the forthcoming sections to derive algorithms more conveniently, and to simplify reductions from and to EEA.

2.2.1.1 The minpath Function

On many occasions we need to find a minimum-weight realization of a path-hint in an advice that starts and terminates in some specified vertices. Hence we need to compute a minimum-weight path that traverses vertices of components in the order given by the hint. The minpath function defined below finds such paths.

Definition 2.2.2. Let the directed multigraph $G = (V, A)$ and the weight function $\omega : V \times V \to [0, \omega_{max}] \cup \{\infty\}$ constitute an instance of EE. Let p be a path in $\mathbb{C}_G$ and let u be a vertex in the component of G that corresponds to the initial vertex of p and v a vertex in the component that corresponds to the terminal vertex of p. Define $\text{minpath}(G, \omega, p, u, v)$ as the shortest path s from u to v in the complete graph $(V, V \times V)$ such that $\mathbb{C}_G(s) = p$.

Recall that we have made shortest-path preprocessing (Transformation 2.1.2) implicit at the start of this section. Thus, by Observation 2.1.4, we may assume

that any shortest path in $(V, V \times V)$ with respect to the weight function ω does not successively visit two vertices of one connected component of G. This gives the following strategy to compute $\text{minpath}(G, \omega, p, u, v)$:

Orient the path p to obtain a directed path p'. Initialize a new weight function ω' that assigns every arc in $V \times V$ the weight ∞. Iterate over the arcs of p'. For any such arc (c_1, c_2) let C_1, C_2 be the corresponding components. For every arc $(w, x) \in C_1 \times C_2$ set $\omega'(w, x) := \omega(w, x)$. Now, using the weight function ω', compute a shortest path s from u to v in the graph $(V, V \times V)$. Return s. See also the pseudocode in Algorithm MinPath.

Algorithm MinPath: Finding minimum-weight paths that traverse components in a specified order.

Input: A directed multigraph $G = (V, A)$, a weight
function $\omega : V \times V \to [0, \omega_{max}] \cup \{\infty\}$, a path p in $\mathbb{C}_G$, and
vertices u, v in the components C^u, C^v corresponding to the initial
and terminal vertices of p, respectively.

Output: A minimum-weight path s from u to v in $(V, V \times V)$ such
that $\mathbb{C}_G(s) = p$.

```
/* Orient the path p.                                            */
```
1 $p' \leftarrow$ a path that is an orientation of p and starts in the vertex
   corresponding to C^u and terminates in the vertex corresponding to C^v;
```
/* Initialize a modified weight function ω'.                     */
```
2 **for** $w, x \in V$ **do** $\omega'(w, x) \leftarrow \infty$;
3 **for** $(c_1, c_2) \in p'$ **do**
4 $\quad$ $C_1 \leftarrow$ connected component of G corresponding to c_1;
5 $\quad$ $C_2 \leftarrow$ connected component of G corresponding to c_2;
6 $\quad$ **for** $w \in C_1, x \in C_2$ **do** $\omega'(w, x) \leftarrow \omega(w, x)$;
7 Path $\leftarrow$ a shortest path from u to v in the complete directed graph with
   the vertices of G and with weight function ω';
8 **return** Path;

Lemma 2.2.1. *Algorithm MinPath computes the function* $\text{minpath}(G, \omega, p, u, v)$ *in* $\mathrm{O}(n^2)$ *time.*

Proof. Consider $p_{min} = \text{minpath}(G, \omega, p, u, v)$. This path retains its weight under the weight function ω'. It follows that the output s of Algorithm MinPath has at most the weight of p_{min}. However, since in any vertex of a component of G only arcs that lead to the next component according to p' may have weight $\leq \infty$, we may assume that $\mathbb{C}_G(s) = p$ and thus $\omega(s) \geq \omega(p_{min})$.

The dominating running time portion is in the computation of a shortest path in line 7, which is possible in $\mathrm{O}(n^2)$ time using Dijkstra's algorithm (there are no negative weights in ω'). $\qquad \square$

Using the minpath function, we can formulate a fact about Eulerian extensions that we use in reductions involving EEA.

Observation 2.2.1. *Let E be an Eulerian extension for the multigraph G that heeds the advice P, let P contain a path-hint h and let ω be a weight*

function $V \times V \to [0, \omega_{max}] \cup \{\infty\}$. There is an Eulerian extension E' such that the following statements hold:

(i) E' heeds the advice P,
(ii) $\omega(E') \leq \omega(E)$, and
(iii) $A(\text{minpath}(G, \omega, h, u, v)) \subseteq E'$.

Here, u, v are vertices contained in the connected components of G that correspond to the initial and terminal vertices of h, respectively.

Proof. Observation 2.2.1 is easy to prove: Simply remove the realization p of h from E and add the edges of $\text{minpath}(G, \omega, h, u, v)$ where u, v are the initial and terminal vertices of p, respectively. □

2.2.1.2   Removing Cycles from an Advice

Now regarding hints to cycles, we may proceed as in Algorithm DetermineCycle (see page 29): First we introduce a new component K' that is a copy of an arbitrary component K visited by the given cycle hint c (lines 1 and 2). Then we extend the weight-function ω such that any arc in $V \times V$ that contains a vertex v of K' is assigned the same weight as the arc that contains the original vertex in K (lines 3 to 5). We then split the cycle c to a path p that goes from K to K' (lines 6 to 9). Then for every vertex $v \in K$ we compute $\text{minpath}(G, \omega, p, v, v')$ and $\text{minpath}(G, \omega, p, v', v)$ where v' is the copy of v in K'. This is done in lines 11 to 18. The shortest path found in this procedure is modified such that the vertex it contains in K' is replaced by its original in K. This modified path is returned.

Lemma 2.2.2. *The output returned by Algorithm DetermineCycle is a cycle that is contained in a minimum-weight Eulerian extension E for G that heeds an advice P such that P contains the input cycle c. The algorithm runs in $O(n^3)$ time.*

Proof. It is easy to see that the output is a cycle: The algorithm computes a path from $v \in K$ to its copy $v' \in K'$. However, v' is replaced by v in the final step in line 19.

Since the Eulerian extension E heeds some advice that contains the cycle-hint c, it contains a number of closed trails that all visit the components whose corresponding vertices in $\mathbb{C}_G$ are contained in c. Let c_{min}^G be a trail that is of minimum-weight among those trails. Because of shortest-path preprocessing and Observation 2.1.4 we may assume that c_{min}^G is a cycle that contains exactly one vertex of every component it visits. By copying an arbitrary component K this cycle visits and modifying the cycle so that it starts in one vertex v of K and ends in the copy of v, we obtain a path of the same weight. That is, the path found by Algorithm DetermineCycle has at most the weight of c_{min}^G. However, it may not find a cycle that is of lower weight than c_{min}^G, otherwise E is not of lowest weight.

Regarding the running time, lines 1 and 2 can be carried out in $O(n+m)$ time. Extending the weight function in lines 3 to 5 is possible in $O(n^2)$ time. Lines 6 to 9 take time at most $O(n)$ using list-implementations of paths. The loop in line 11 is executed at most n times and every iteration takes $O(n^2)$ time using Algorithm MinPath. Summing up, we get a bound of $O(n^3)$ time. □

Lemma 2.2.2 yields the following theorem:

Algorithm DetermineCycle: Finding minimum-weight cycles with advice.

    Input: A directed multigraph $G = (V, A)$, a weight
           function $\omega : V \times V \rightarrow [0, \omega_{max}] \cup \{\infty\}$ and a cycle c in $\mathbb{C}_G$.
    Output: A minimum-weight cycle in G that occurs in an Eulerian
           extension of G that heeds an advice containing c.

    /* Introduce a new component to split the cycle.   */
1 $K \leftarrow$ an arbitrary component of G that is visited by c;
2 $G \leftarrow G$ with an additional copy K' of K;
3 **for** $(v, w) \in K \times V$ **do**
4 $v' \leftarrow$ the copy of v in K';
5 $\omega(v', w) \leftarrow \omega(v, w)$;

6 $k \leftarrow$ the vertex in $\mathbb{C}_G$ that corresponds to K;
7 $k' \leftarrow$ the vertex in $\mathbb{C}_G$ that corresponds to K';
8 $\{k, v\} \leftarrow$ an edge in c that is incident to k;
9 $p \leftarrow c \setminus (\{\{k, v\}\} \cup \{\{k', v\}\})$;
    /* Probe vertices for shortest cycles.   */
10 CurrentShortestPath $\leftarrow$ empty list;
11 **for** $v \in K$ **do**
12 $v' \leftarrow$ the copy of v in K';
13   Path $\leftarrow$ minpath(G, ω, p, v, v');
14   Path$' \leftarrow$ minpath(G, ω, p, v', v);
15   **if** $\omega'(\text{Path}) < \omega'(\text{CurrentShortestPath})$ **then**
16   CurrentShortestPath $\leftarrow$ Path;
17   **if** $\omega'(\text{Path}') < \omega'(\text{CurrentShortestPath})$ **then**
18   CurrentShortestPath $\leftarrow$ Path$'$;

19 **return** CurrentShortestPath *with every vertex in* CurrentShortestPath $\cap K'$
    replaced by its original in K;

Theorem 2.2.1. EULERIAN EXTENSION WITH ADVICE *and* EULERIAN EXTENSION WITH CYCLE-FREE ADVICE *are equivalent under polynomial-parameter polynomial-time many-one reductions when parameterized by the number of connected components and/or the sum of positive balances of all vertices.*

Proof. Since EE∅A is a subset of EEA this direction is trivial. To reduce EEA to EE∅A simply use Algorithm DetermineCycle for every cycle-hint in the advice and add the corresponding cycle to the input graph. This is a polynomial-time many-one reduction, because it can be carried out in $O(|P|n^3)$ time and it is correct because of Lemma 2.2.2. Also, by carrying out the reduction the number of components does not increase and the balance of all vertices stays the same. As a consequence, this is a polynomial-parameter polynomial-time reduction for these parameters. □

Theorem 2.2.1 enables us to simplify reductions and algorithms for EEA by using the equivalence of EEA and EE∅A and by considering the simpler problem of EE∅A instead.

2.2.2 The Impact of Advice

In this section we investigate the relationship of EE and EEA. For this, we consider the following restricted form of advice and corresponding problem EULERIAN EXTENSION WITH MINIMAL CONNECTING ADVICE (EECA).

Definition 2.2.3. Let G be a directed multigraph and let P be an advice for G. We call the advice P *connecting*, if the hints in P connect every vertex in $\mathbb{C}_G$.

> EULERIAN EXTENSION WITH MINIMAL CONNECTING ADVICE
> *Input:* A directed multigraph $G = (V, A)$ with a weight function $\omega :$
> $V \times V \to [0, \omega_{max}] \cup \{\infty\}$ and minimal connecting advice P.
> *Question:* Is there an Eulerian extension E of G that is of weight at
> most ω_{max} and heeds the advice P?

We show that EE is parameterized Turing-reducible to EECA when parameterized by the number c of components in the input graph or the combined parameter of c and the sum b of all positive balances of vertices in the input graph. And we also give a polynomial-time polynomial-parameter many-one reduction from EEA to EE with respect to the parameter number of connected components in this section.

Since in Chapter 3 we will show that a polynomial-size problem kernel for EE would imply coNP ⊆ NP/poly and since in Subsection 2.3.2 we will give a polynomial-size problem kernel for EECA, we cannot hope to replace the Turing reduction with a polynomial-time polynomial-parameter many-one reduction. Otherwise we could derive a polynomial-size problem kernel for EE using this reduction.

In terms of classical complexity theory, the parameterized Turing reduction is a very powerful tool, and thus, one could hope for EECA being polynomial-time solvable. This, however, is unlikely. Although the reductions given in this section do not imply a hardness result for EECA, we gather NP-hardness as a simple corollary (Corollary 2.3.5) in Subsection 2.3.2. Nevertheless, the reductions given in this section are of high value to us, because we can use the Turing reduction to derive an efficient algorithm for EE in Subsection 2.2.3 and together with the second reduction, we can restate EE as a matching problem in Section 2.3.

Simple Observations Regarding EECA. For running time analysis, we sometimes need to know the maximum number of hints in an advice in EECA. Here, the following observation is helpful.

Observation 2.2.2. *Let G be a directed multigraph with c connected components and let P be a minimal connecting advice for G. The advice P contains at most c hints.*

Proof. Since a hint is a path or cycle of length at least one, it connects at least two vertices in $\mathbb{C}_G$. We consider the graph $(V(\mathbb{C}_G), \varnothing)$ and the procedure of successively adding hints $h_1, \ldots, h_k$ that form a minimal connecting advice. It is clear that every hint $h_i, 1 \leq i \leq k$, must connect two connected components of the graph $(V(\mathbb{C}_G), \bigcup_{j=1}^{i-1} E(h_j))$. Otherwise we could remove h_i and still connect every vertex in $\mathbb{C}_G$ using the remaining hints. Thus, adding c hints connects every vertex in $\mathbb{C}_G$ and there are at most c hints in P. $\square$

It is also easy to see, that we can realize every cycle hint in a minimal connecting advice to obtain a cycle-free minimal connecting advice.

Observation 2.2.3. EULERIAN EXTENSION WITH MINIMAL CONNECTING ADVICE *is equivalent to* EULERIAN EXTENSION WITH CYCLE-FREE MINIMAL CONNECTING ADVICE *(*EE∅CA*) under polynomial-parameter polynomial-time many-one reductions with respect to the parameters number of connected components and sum of all positive balances of vertices.*

Proof. See Theorem 2.2.1. $\square$

2.2.2.1 Reducing EE to EECA

To reduce EE to EECA the obvious idea of trying pieces of advice yields a Turing reduction. We make use of the observations in Section 2.1 to assume certain restrictions on the pieces of advice we have to guess.

Lemma 2.2.3. *Let G be a directed multigraph and let E be a minimum-weight Eulerian extension with respect to a weight function $\omega : V \times V \to [0, \omega_{max}] \cup \{\infty\}$ for G. There is a minimal connecting advice $P = \{h_1, \ldots, h_i\}$ such that*
  (i) E heeds P, and
  (ii) the graph defined by the union of all trails $h_1, \ldots, h_i$ without their initial vertices does not contain a cycle.

Proof. This is mainly based on Theorem 2.1.1. By the theorem, there is a decomposition of E into paths and cycles $t_1, \ldots, t_k$ such that the graph defined by the union of all trails $\mathbb{C}_G(t_1), \ldots, \mathbb{C}_G(t_k)$ without their initial vertices does not contain a cycle. We greedily take paths $\mathbb{C}_G(t_j)$ of length at least one into P that connect new vertices in $\mathbb{C}_G$. $\square$

Using this restriction, we can guess all forests of $\mathbb{C}_G$ and try all possibilities to extend them to an advice:

Lemma 2.2.4. EULERIAN EXTENSION *is parameterized Turing-reducible to* EULERIAN EXTENSION WITH MINIMAL CONNECTING ADVICE *when parameterized by the number c of components in the input graph or the combined parameter of c and the sum of all positive balances of vertices in the input graph. The reduction can be carried out in $O(16^{c \log(c)}(c + n + m))$ time.*

Proof. Let the directed multigraph $G = (V, A)$ and the weight function $\omega : V \times V \to [0, \omega_{max}] \cup \{\infty\}$ constitute an instance of EE and let c be the number of connected components in G. We give an algorithm that decides EE using an oracle for EECA in time $O(2^{c^2 \log(c)}(c^3 + n + m))$.

We simply generate all possible pieces of advice and apply the oracle to the resulting instances. If one of the oracle calls accepts the advice-instance, then, clearly, the original instance is a yes-instance. Also, for every yes-instance of EE, there is an advice derivable from a solution to the instance because of Lemma 2.2.3. Clearly, the number of components and the sum of all positive balances remain the same in the instances passed to the oracle.

Concerning the generation of the pieces of advice, by Lemma 2.2.3 we may assume that the hints without their initial vertices form a forest in $\mathbb{C}_G$. Thus, we may simply enumerate all forests contained in $\mathbb{C}_G$, partition their edges into at most c hints and try all possibilities to add the initial vertex back onto the hints.

To enumerate all forests, we first partition the vertices into at most c cells (there are c^c many such partitions), then enumerate all spanning trees in each cell (in each cell there are c^{c-2} spanning trees [5]). This is possible in $O(c^c(c^{c-2} + c^2)) = O(c^{2c-2})$ time [22].

We then partition the edges into at most c hints (there are c^c partitions), extend every hint by adding an initial vertex (in total, there are c^c possibilities) and check if this yields a valid advice—that is, whether the hints are paths or cycles and whether the advice is connecting. This procedure can be carried out in $O(c^{2c}c^3)$ time allowing $O(c^3)$ for the validity check.

For every guessed advice, we have to pass the instance to the oracle in linear time and, since $n^n = 2^{n \log(n)}$, we can derive the running time bound of $O(16^{c \log(c)}(c + n + m))$. $\qquad\square$

2.2.2.2 Reducing EEA to EE

Here, we will see that there is only a polynomial number of optimal ways to realize a hint in an advice. Each of these realizations will be modeled by a pair of imbalanced vertices. These pairs will reside in a new component and this component then can only be connected to the rest of the graph by taking arcs into an Eulerian extension that also connect each component corresponding to inner vertices of the hint.

For convenience, we give a reduction from EE$\varnothing$A (see Subsection 2.2.1) instead of EEA. This is without loss of generality because of Theorem 2.2.1. We first give an intuitive description, followed by detailed construction and then a correctness proof. The construction uses the minpath function introduced on page 26 in Subsection 2.2.1.

Intuitive Description. We look at the hints present in an EE$\varnothing$A instance and eliminate them one at a time: For every hint p_i, $1 \le i \le d$, first, a connected component is introduced (vertex set W_1^i, arc sets $B_1^{i,\pm}, B_1^{i,=}$ in the construction below) and copied for every inner vertex of the hint (vertex sets W_l^i, arc sets $B_l^{i,\pm}, B_l^{i,=}$ for $2 \le l \le k-1$). Each copy is connected to the component corresponding to its vertex in the hint (by the arc-set $B_l^{i,\gamma}$). The new component and its copies consist of interconnected imbalanced pairs of vertices. In the

construction below, these are the vertices $s^{i,\pm}_{l,u,v}, t^{i,\pm}_{l,u,v}$ contained in the i-th component. Each pair corresponds to a pair of vertices u, v forming the endpoints of a path that realizes the currently considered hint p_i.

A new weight function gives meaning to the construction and ensures that adding an arc $(u, t^{i,+}_{1,u,v})$ or an arc $(s^{i,-}_{1,u,v}, v)$ to an Eulerian extension has the same weight as a minimum-weight realization of the hint that goes from u to v or from v to u, respectively. Notice that the superscript "+"corresponds to paths in one direction and the superscript "−" to paths in the opposite direction. The weight function also ensures that if such an arc is present in an Eulerian extension, then the connected components traversed by the hint are connected to each other.

Construction 2.2.1. Let the directed multigraph $G_0 = (V_0, A_0)$, the weight-function $\omega_0 : V_0 \times V_0 \to [0, \omega_{max}] \cup \{\infty\}$, and the advice P constitute an instance $I_{EE\varnothing A}$ of EE∅A. Let $p_1, \ldots, p_d$ be the elements of P and let $C_1, \ldots, C_c$ be the connected components of G.

For every $p_i, 1 \leq i \leq d$, inductively define G_i and ω_i as follows: Let $C_{j_1}, \ldots, C_{j_k}$ be the components of G that correspond to the vertices traversed by p_i, ordered according to an arbitrary path orientation of p_i. For every $1 \leq l \leq k-1$ introduce the vertex set

$$W^{i,+}_l := \{t^{i,+}_{l,u,v}, s^{i,+}_{l,u,v} : u \in C_{j_1} \cap I^+_G \wedge v \in C_{j_k} \cap I^-_G\}, \text{ and}$$
$$W^{i,-}_l := \{s^{i,-}_{l,u,v}, t^{i,-}_{l,u,v} : u \in C_{j_1} \cap I^-_G \wedge v \in C_{j_k} \cap I^+_G\}.$$

Set $W^i_l := W^{i,+}_l \cup W^{i,-}_l$. Make all these vertices imbalanced via the arc set

$$B^{i,\pm}_l := \{(t^{i,+}_{l,u,v}, s^{i,+}_{l,u,v}), (t^{i,-}_{l,u,v}, s^{i,-}_{l,u,v})\}.$$

Let $w^1_l, \ldots, w^h_l$ be the vertices in W^i_l. For each $1 \leq l \leq k - 1$, interconnect these vertices via a cycle, using the following arc set

$$B^{i,=}_l := \{(w^g_l, w^{g+1}_l) : 1 \leq g < h\} \cup \{(w^h_l, w^1_l)\}.$$

Furthermore, for each $2 \leq l \leq k - 1$, choose $c_{j_l} \in C_{j_l}$ and $w_l \in W^i_l$ arbitrarily and add the following arc set connecting W^i_l to C_{j_l}:

$$B^{i,\gamma}_l := \{(w_l, c_{j_l}), (c_{j_l}, w_l)\}.$$

Now set $G_i = (V_i, A_i) := (V_{i-1} \cup \bigcup_{l=1}^{k-1} W^i_l, A_{i-1} \cup \bigcup_{l=1}^{k-1}(B^{i,\pm}_l \cup B^{i,=}_l) \cup \bigcup_{l=2}^{k-1} B^{i,\gamma}_l))$ and create a new weight function as follows:

$$\omega_i(u,v) := \begin{cases} \omega_{i-1}(u,v), & u, v \in V_{i-1} \\ \omega_0(\text{minpath}(G_0, \omega_0, p_i, u, x)), & u \in C_{j_1} \cap I^+_G, v = t^{i,+}_{1,u,x} \\ \omega_0(\text{minpath}(G_0, \omega_0, p_i, x, v)), & u = s^{i,-}_{1,x,v}, v \in C_{j_1} \cap I^-_G \\ 0, & u = s^{i,+}_{k-1,x,v}, v \in C_{j_k} \cap I^-_G \\ 0, & u \in C_{j_k} \cap I^+_G, v = t^{i,-}_{k-1,u,x} \\ 0, & u = s^{i,\pm}_{l,x,y}, v = t^{i,\pm}_{l,x,y} \\ 0, & u = s^{i,\pm}_{l,x,y}, v = t^{i,\pm}_{l+1,x,y} \\ \infty, & \text{otherwise} \end{cases}$$

The graph G_d, the weight function ω_d and the number ω_{max} constitute an instance I_{EE} of EE.
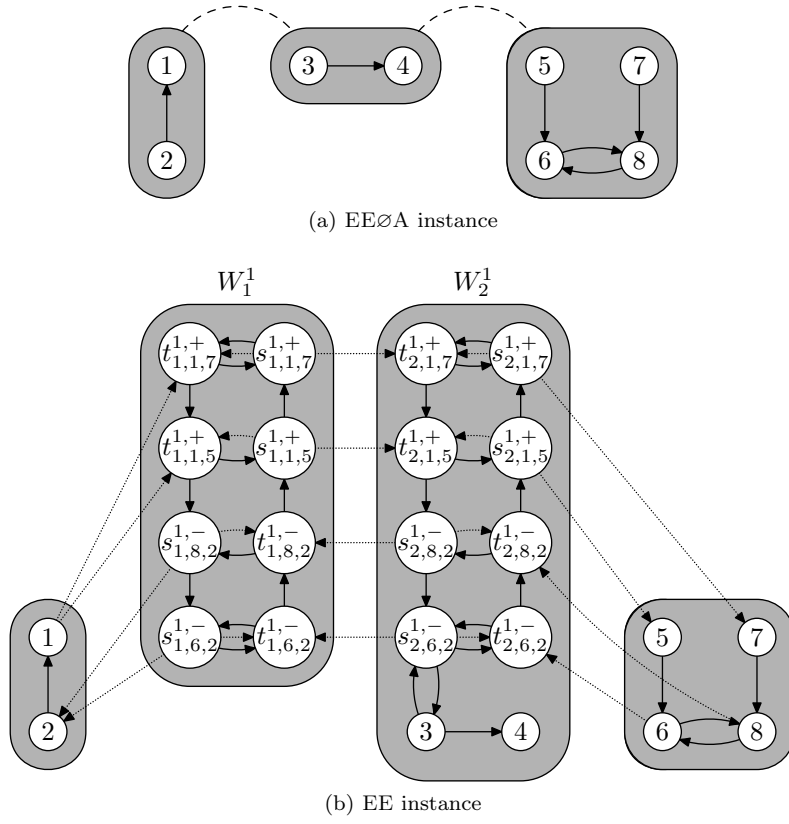
(a) EE∅A instance

(b) EE instance

Figure 2.6: Example for Construction 2.2.1 explained in Example 2.2.1.

Example 2.2.1. Have a look at Figure 2.6. At the top, an instance $I_{\text{EE}\varnothing\text{A}}$ of EE∅A is shown. It comprises three connected components and an advice consisting of a single hint p_1 represented by the dashed edges. Below, there is an instance I_{EE} of EE produced by Construction 2.2.1. The dotted arcs represent the only arcs incident to the new vertices with weight potentially lower than ∞.

In the new instance the hint p_1 is removed and a new component W_1^1 is introduced. A copy W_2^1 of the vertex set W_1^1 is introduced and connected to the component that corresponds to the inner vertex of p_1. The induced subgraphs of W_1^1, W_2^1 consist of pairs $t_{l,u,v}^{i,+}, s_{l,u,v}^{i,+}$ of vertices that are made imbalanced via a direct arc and that are connected via a directed cycle. Each of the vertices $s_{l,u,v}^{i,+}$—the "sources"—has balance 1 and can either be connected to a vertex $t_{l,u,v}^{i,+}$—the "targets"—inside the same component or to another component. Analogously, targets can only accept at most one arc from either inside the same component or from outside.

Consider a solution E to $I_{\text{EE}\varnothing\text{A}}$ that also contains the arcs $(1,3),(3,5)$ as realization of p_1. We may remove these arcs and add the arcs

$$(1, t_{1,1,5}^{1,+}), (s_{1,1,5}^{1,+}, t_{2,1,5}^{1,+}), (s_{2,1,5}^{1,+}, 5)$$

to E, and add arcs from all remaining sources to their corresponding targets that reside in the same component to obtain a solution to I_{EE}. Also, every solution to I_{EE} has to connect the connected component W_1^1 to the rest of the graph. This is only possible by adding an arc from a source to outside its component, for example at $s_{1,6,2}^{1,-}$. Then the vertex $t_{1,6,2}^{1,-}$ has to fetch an arc from $s_{2,6,2}^{1,-}$ in the Eulerian extension in order to become balanced. This means that then also the arc $(6, t_{2,6,2}^{1,-})$ has to be included in an Eulerian extension for $I_{\text{EE}\varnothing\text{A}}$ and thus we can include the path from vertex 6 to vertex 2 that realizes p_1 computed by the minpath function.

Correctness. We first prove that Construction 2.2.1 is polynomial-time computable and that the parameter in the reduced instance is polynomial in the original parameter. We then proceed to show the soundness of the construction.

Observation 2.2.4. *Construction 2.2.1 is polynomial-time computable. There are $\mathrm{O}(c^2)$ components in G_d.*

Proof. We first look at the running time of the construction: The size of W_l^i and the arc sets $B_l^{i,\pm}, B_l^{i,=}, B_l^{i,\gamma}$ is at most $\mathrm{O}(n^2)$. It holds that $l \leq c$ and there are at most $\mathrm{O}(c^2)$ hints in an advice (recall that hints in an advice are edge-disjoint). Hence, at most $\mathrm{O}(c^3 n^2)$ vertices and edges are added. This can be done in time linear in the number of added vertices and edges. Thus, the new weight-function can be computed in $\mathrm{O}(c^6 n^4)$ time and this yields a polynomial-time algorithm for Construction 2.2.1.

Since there are at most $\mathrm{O}(c^2)$ hints in an advice and for every hint, there is exactly one new component (the component with vertex-set W_1^i) in the reduced instance, the new parameter is in $\mathrm{O}(c^2)$. □

Lemma 2.2.5. *Construction 2.2.1 is a polynomial-parameter polynomial-time reduction.*

Proof. By Observation 2.2.4 it only remains to show that Construction 2.2.1 is correct. For this, first consider an Eulerian extension E that is a solution to $I_{\text{EE}\varnothing\text{A}}$. For every hint p_i the set E contains a set of paths that realize that hint. Without loss of generality we may assume that among those paths is $s = \text{minpath}(G_0, \omega_0, p_i, u, v)$ for suitable vertices u, v in the components that p_i starts and ends, respectively (see Observation 2.2.1). Thus, in order to connect the component W_l^i to the rest of the graph, we may remove s from E and add the arcs

$$(u, t_{1,u,v}^{i,+}), (s_{1,u,v}^{i,+}, t_{2,u,v}^{i,+}), \ldots, (s_{k-2,u,v}^{i,+}, t_{k-1,u,v}^{i,+}), (s_{k-1,u,v}^{i,+}, v).$$

This does not increase the weight of E. To balance all vertices $t_{l,u',v'}^{i,+}, s_{l,u',v'}^{i,+}$ with $1 \leq l \leq k-1, u' \neq u, v' \neq v$, we may add the corresponding arcs $(s_{l,u',v'}^{i,+}, t_{l,u',v'}^{i,-})$ and analogously for vertices in $W_l^{i,-}$, again without increasing the weight. Thus, doing this for every hint yields an Eulerian extension for I_{EE} of the same weight.

Now consider an Eulerian extension E that is a solution to I_{EE}. The set E has to connect the component W_1^i to the rest of the graph for every hint p_i. Thus, without limitation of generality, there is an arc $(u, t_{1,u,v}^{i,+})$ for some vertices u, v in the components that correspond to the endpoints of p_i. For every vertex $t_{l,x,y}^{j,\pm}$ all arcs with weight lower than ∞ end in it, and since it has balance -1, there is exactly one arc incident to it in E. The same is true for vertices $s_{l,x,y}^{j,\pm}$ since all arcs with weight lower than ∞ start at them and they have balance 1. Hence the arc $(s_{1,u,v}^{i,+}, t_{2,u,v}^{i,+})$ is present in E, by induction also $(s_{l,u,v}^{i,+}, t_{l+1,u,v}^{i,+}) \in E, 1 \leq l \leq k-2$, and finally also $(s_{k-1,u,v}^{i,+}, v) \in E$. Thus we can remove these arcs from E, add $\text{minpath}(G_0, \omega_0, p_i, u, v)$, and repeat this for all hints to obtain an Eulerian extension for G_0 that heeds the advice P and has weight at most w_{max}. $\square$

Theorem 2.2.2. EULERIAN EXTENSION WITH ADVICE *is polynomial-time polynomial-parameter many-one reducible to* EULERIAN EXTENSION *when parameterized by the number of components in the input graph.*

2.2.3 An Efficient Multivariate Algorithm for EECA

In this section we consider EECA parameterized by both the number of components c in the input graph and the sum b of all positive balances of vertices in the input graph. A simple idea is used to obtain an efficient algorithm that solves EECA. We prove the following theorem:

Theorem 2.2.3. EULERIAN EXTENSION WITH MINIMAL CONNECTING ADVICE *is solvable in* $O(4^{c \log(b)} n^2 (b^2 + n \log(n)) + n^2 m)$ *time, where c is the number of components in the input graph and where b is the sum of all positive balances of vertices in the input graph.*

In a simple corollary, we also obtain an efficient algorithm for EE, proving that this problem is fixed-parameter tractable with respect to the combined parameter (b, c). We deem parameterizing with both b and c to be a good choice: The reduction we use to show NP-hardness for EE in Subsection 1.2.2 creates instances where $b = 0$ implying that parameterizing only with b does not suffice to obtain efficient algorithms. Also, the question whether EE is fixed-parameter tractable with respect to parameter c is a long-standing open question dating

back to Frederickson [18]. We reflect on the parameter c in Section 2.3 and it seems hard to answer this question.

To obtain an algorithm for EECA, we use the fact that minimum-weight Eulerian extensions for connected multigraphs can be found in $O(n^3 \log(n))$ time [10]. To derive a connected instance of EE from an instance of EECA, we realize all hints in the given minimal connecting advice. The parameter b helps to bound the number of possible ways we have to try to realize each hint. An algorithm that achieves the running time given in Theorem 2.2.3 can simply try each combination of optimal realizations of each hint in the given advice and then solve the resulting instance comprising a connected multigraph via the polynomial-time algorithm given by Dorn et al. [10]. We denote a call to this algorithm by solve_connected(G, ω), where G is a connected multigraph and $\omega : V \times V \to [0, \omega_{max}] \cup \{\infty\}$ is a weight function.

Solution Algorithm. For convenience, we give an algorithm that solves EE∅CA which we then generalize to an algorithm for EECA using Observation 2.2.3. In Algorithm SolveEE∅CA a description of the solution algorithm is shown in pseudo code. It is invoked with an instance of EE∅CA and an empty set E. The set E is then successively extended to a minimum-weight Eulerian extension. This is done by iterating over every local-optimal realization of each hint in lines 9 and 10 and recursing for every of them. When each hint is realized, that is $P = \emptyset$ in line 1, the resulting instance is solved in polynomial time.

Algorithm SolveEE∅CA: Solving EE∅CA.

   Input: A directed multigraph $G = (V, A)$, a weight
        function $\omega : V \times V \to [0, \omega_{max}] \cup \{\infty\}$, a cycle-less advice P and
        an arc-set E.
   Output: A minimum-weight Eulerian extension for G that heeds the
        advice P.

1   **if** $P = \emptyset$ **then**
2   **return** $E \cup$ solve_connected(G, ω);
3   **else**
4 $h \leftarrow$ a hint in P;
5 $v_A \leftarrow$ initial vertex of h;
6 $C_A \leftarrow$ connected component of G corresponding to v_A;
7 $v_\Omega \leftarrow$ terminal vertex of h;
8 $C_\Omega \leftarrow$ connected component of G corresponding to v_Ω;
9   MinEE $\leftarrow \emptyset$;
10   found_solution $\leftarrow$ false;
11   **for** $(u, v) \in I_G^+ \times I_G^-$ *such that* $u \in C_A \wedge v \in C_\Omega$ *or vice versa* **do**
12 $p \leftarrow$ minpath(G, ω, h, u, v);
13   ActEE $\leftarrow$ SolveEE∅CA$(G + p, \omega, P \setminus \{h\}, E \cup p)$;
14   **if** $\omega(\text{MinEE}) > \omega(\text{ActEE}) \vee$ found_solution $=$ false **then**
15   found_solution $\leftarrow$ true;
16   MinEE $\leftarrow$ ActEE;
17   **return** MinEE;

Proof of Theorem 2.2.3. The theorem is mainly based on Algorithm Solve-EE∅CA: Given an instance of EECA we compute an equivalent instance of EE∅CA using the reduction in Theorem 2.2.1 that uses Algorithm Determine-Cycle. Then, we apply Algorithm SolveEE∅CA solving the instance of EE∅A. We first look at the correctness of Algorithm SolveEE∅CA and then analyze the overall running time.

Consider the return value E' of Algorithm SolveEE∅CA when called with an initially empty arc set E and an instance of EE∅A consisting of the multigraph G, the weight function ω, and minimal connecting advice P. For every hint in P there is realization in E', that is, E' connects all connected components of G. Because of the call to solve_connected the set E' also makes every vertex in G balanced. Hence E' is an Eulerian extension for G that heeds P. Also, E' is of minimum weight among all Eulerian extensions for G that heed the advice P, because of the weight-minimality of solve_connected and because, by Observation 2.2.1, we may assume that in a minimum-weight Eulerian extension all path hints h are realized by $\mathrm{minpath}(G, \omega, h, u, v)$ for appropriate vertices u, v in the components of G corresponding to the initial and terminal vertices of h.

Concerning the running time of the overall procedure, we have to preprocess the input instance using Transformation 2.1.1 and Transformation 2.1.2 (we have made this preprocessing implicit at the start of the section). By Lemmas 2.1.1 and 2.1.2 this takes $\mathrm{O}(n^3 + n^2 m)$ time. Next, the given instance of EECA has to be converted to an instance of EE∅CA. By Lemma 2.2.2 this is possible in $\mathrm{O}(|P|n^3)$ time. Finally, we apply Algorithm SolveEE∅CA: Obviously its recursion depth is at most $|P|$. Because of $b \geq |I_G^+| = |I_G^-|$, every call of Algorithm SolveEE∅CA yields at most b^2 recursive calls. This means the sum of all calls is $b^{2|P|}$. The running-time of one call is dominated by either the computation of b^2 minpath-instances which takes $\mathrm{O}(b^2 n^2)$ time (Lemma 2.2.1) or the computation of solve_connected which takes $\mathrm{O}(n^3 \log(n))$ time [10]. Thus, Algorithm SolveEE∅CA can be computed in

$$\mathrm{O}(b^{2|P|}(b^2 n^2 + n^3 \log(n))) = \mathrm{O}(2^{2|P|\log(b)} n^2 (b^2 + n \log(n)))\ \text{time}.$$

Because of Observation 2.2.2, $|P| \leq c$ and thus we can derive that the running-time bound of the overall procedure is in

$$\mathrm{O}(2^{2c\log(b)} n^2 (b^2 + n \log(n)) + cn^3 + n^2 m)$$
$$\subseteq \mathrm{O}(4^{c\log(b)} n^2 (b^2 + n \log(n)) + n^2 m). \qquad \square$$

Corollary 2.2.1. EULERIAN EXTENSION *is solvable in*

$$\mathrm{O}(4^{c\log(bc^2)} n^2 (b^2 + n \log(n)) + n^2 m)\ \text{time.}$$

Proof. By Lemma 2.2.4 there is a Turing reduction from EE to EECA with running time of $\mathrm{O}(16^{c\log(c)}(c + n + m))$ and at most $16^{c\log(c)}$ oracle calls. Replacing the oracle with the algorithm for EECA given in Theorem 2.2.3 we obtain an algorithm for EE with $\mathrm{O}(4^{c\log(bc^2)} n^2 (b^2 + n \log(n)) + n^2 m)$ running time: The algorithm first preprocesses the input using Transformation 2.1.1 and Transformation 2.1.2, guesses the advice and then, instead of invoking the oracle, reduces the resulting instance of EECA to an instance of EE∅CA. This instance is then solved using Algorithm SolveEE∅CA. $\square$

2.3 From Eulerian Extension to Matching and back

The observations in Section 2.1 suggest the following intuition for making multigraphs Eulerian: To balance every vertex in the given multigraph, we have to add paths from vertices with lower outdegree to vertices with lower indegree. This implies that we have to match these vertices such that adding paths between them leads to a minimum-size Eulerian extension. In this section we prove that this intuition is correct and restate EE as the newly introduced CONJOINING BIPARTITE MATCHING (CBM).

In previous work by Dorn et al. [10] a similar approach that involves matchings yields polynomial-time algorithms for some restricted Eulerian extension problems. Of course polynomial-time solvability would be very surprising for EE because this problem is NP-hard; and we will see that the corresponding matching problem CBM indeed is also NP-hard. However, we deem the matching representation to be more accessible in terms of fixed-parameter complexity. In this regard, we show that CBM is fixed-parameter tractable on restricted input graphs for a parameter that translates over to the number of components in EE. Using this we make partial progress to answering the question whether EE is fixed-parameter tractable with the parameter number of connected components by showing that it indeed is fixed-parameter tractable in a restricted form. We also gather a polynomial-size problem kernel for EECA as a simple corollary using the matching formulation.

In Subsection 2.3.1 we introduce CBM, show that it is NP-hard, and derive that it is fixed-parameter tractable on special input graphs. In Subsection 2.3.2 we investigate the relationship between EE and CBM, and show that they are parameterized equivalent. Using this equivalence, we derive fixed-parameter tractability results for EE as simple corollaries.

2.3.1 Conjoining Bipartite Matching

In this section we introduce CONJOINING BIPARTITE MATCHING (CBM)—a variant of minimum-weight perfect bipartite matching. We show that this problem is NP-hard and fixed-parameter tractable on a restricted graph class.

Definition 2.3.1. Let G be a bipartite graph,[3] let M be a matching of the vertices in G and let P be a vertex-partition with the cells $C_1, \ldots, C_c$. We call an unordered pair $\{i, j\}$ of integers $1 \leq i < j \leq c$ a *join* and a set J a *join set* with respect to G and P if $J \subseteq \{\{i, j\} : 1 \leq i < j \leq c\}$. We say that a join $\{i, j\} \in J$ is *satisfied* by the matching M of G if there is at least one edge $e \in M$ with $e \cap C_i \neq \varnothing$ and $e \cap C_j \neq \varnothing$. We say that a matching M of G is *J-conjoining* with respect to a join set J if all joins in J are satisfied by M. If the join set is clear from the context, we simply say that M is conjoining.

> CONJOINING BIPARTITE MATCHING (CBM)
> *Input:* A bipartite graph $G = (V_1 \uplus V_2, E)$ with a weight function $\omega : E \to [0, \omega_{max}] \cup \{\infty\}$, a partition $P = \{C_1, \ldots, C_k\}$ of the vertices in G and a join set J.
> *Question:* Is there a matching M of the vertices of G such that M is perfect, M is conjoining and M has weight at most ω_{max}?
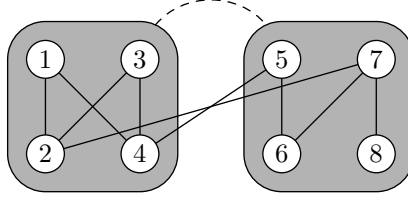
---

[3]Note that G is undirected.

Figure 2.7: An instance of CBM comprising a bipartite graph with the vertices 1 through 8 and the solid edges, a vertex partition represented by the gray objects, and a join set consisting of a single join that is represented by the dashed line. The weight function is ignored here. The shown instance has a perfect matching, for example $\{\{1,2\},\{3,4\},\{5,6\},\{7,8\}\}$. However, it does not have a perfect and conjoining matching: The vertex 8 has to be matched to 7 in any perfect matching. Thus, the vertices 2 and 7 cannot be matched. Since 7 is already matched, the vertex 6 has to be matched to 5. This means that the vertices 4 and 5 cannot be matched. Thus, no edge that satisfies the single join present can be contained in a perfect matching.

For an example of an instance of CBM, see Figure 2.7.

Example 2.3.1. CBM models a variant of the assignment problem with additional constraints. In this variant, an assignment of workers to tasks is sought such that each worker is busy and each task is being processed. Furthermore, every worker must be qualified for its assigned task. Both the workers and the tasks are grouped and the additional constraints are of the form "At least one worker from group A must be assigned a task in group B". An assignment that satisfies such additional constraints may be favorable in the following scenario.

A company wants to create working groups, each working on a distinct project consisting of multiple tasks. However, every working group shall have a very creative member, a very social and a very methodical member. Here, we assume that extreme creativity, sociality and methodicality are mutually exclusive.

This scenario can be modeled in CBM by defining a bipartite graph that has a vertex for every worker and task, and that has an edge between a worker and a task, if the worker is qualified for the task. The additional constraints can be modeled by first partitioning the tasks into the projects $C_1, \ldots, C_i$ and partitioning the workers into the creative ones C_{i+1}, the social ones C_{i+2} and the methodical ones C_{i+3}. Then, creating a join set $\{\{j, i+1\}, \{j, i+2\}, \{j, i+3\} : 1 \leq j \leq i\}$ ensures that every working group is assigned at least one creative, social, and methodical member.

The edge weights can be ignored in our scenario. However, as we will see in the forthcoming section, the problem of CBM is NP-hard even in the unweighted case.

2.3.1.1   NP-Hardness

We reduce from the well-known 3SAT problem [23]. For this, we briefly recapitulate some related definitions.

Definition 2.3.2. Consider the boolean variables $X = \{x_1, \ldots, x_n\}$. *Positive literals* over X are x_i and *negative literals* are $\neg x_i$ with $x_i \in X$. A *boolean*

formula ϕ in conjunctive normal form over the variables X is of the form $\bigwedge_{i=1}^{k} c_i$, where $c_i = l_{i_1} \vee \ldots \vee l_{i_{j_i}}$. Here l_i, $1 \leq i \leq 2n$ are literals over X. The subformulas c_i, $1 \leq i \leq k$, are called *clauses*. If it holds that $j_1 = \ldots = j_k = d$, then we say that ϕ is in *d-conjunctive normal form*. A *truth assignment* ν for the variables X is a function $\nu : X \rightarrow \{\text{true}, \text{false}\}$. A truth assignment is said to be *satisfying* for a boolean formula ϕ if ϕ evaluates to true when substituting $\nu(x_i)$ for every variable x_i occuring in ϕ.

In 3SAT, a boolean formula ϕ in 3-conjunctive normal form is given and it is asked whether there is a truth-assignment of the variables in ϕ that satisfies ϕ. We use the fact that, in CBM, connected components that form cycles have exactly two perfect matchings because every cycle in a bipartite graph has even length. Thus, we model variables as cyclic connected components and the two possible matchings will correspond to the two possible truth values for the variables. Clauses will be modeled by cells in the input partition and a join that forces one of the corresponding variable-cycles into one of the two possible matchings in order to satisfy the clause.

In the following, we regard clauses of boolean formulas in 3-conjunctive normal form over the variables X as subsets of $X \times \{+, -\}$ where $(x_i, +)$ $((x_i, -))$ in the clause c_j implies that x_i is in the clause c_j as a positive (negative) literal.

First, we give an intuitive description of the reduction, we then go into the details. After that, we give an example and prove the correctness of the reduction.

Intuitive Description. Let ϕ be a boolean formula in 3-conjunctive normal form with n variables and m clauses. For every variable x_i we introduce a cycle consisting of $4m$ vertices (vertex set V_i and edge set E_i in the below construction). For every such cycle, we fix an ordering of the edges $e_i^1, \ldots, e_i^{4m}$ according to the order in which they are traversed by the cycle. In a perfect matching of the cycle either all edges e_i^k with odd k are matching edges or all edges e_i^k with even k are matching edges. These two matchings will correspond to assigning x_i the value false or true, respectively.

Next, for every clause c_j we define a cell C_j in order to derive a partition of the vertices in the cycles. For every positive literal $(x_i, +)$ contained in c_j, we choose an edge e_i^k such that k is even and such that its vertices have not been assigned to a cell yet, and put both endpoints of e_i^k into C_j. Analogously, for every negative literal $(x_i, -) \in c_i$ we choose an edge e_i^k such that k is odd and such that its endpoints have not been assigned yet, and put them into C_j. Finally, all vertices that have not been assigned to a cell yet, are added to the cell C_0 and for every cell $C_i, i \geq 1$ we add the join $\{0, i\}$ to the designated join set.

Construction 2.3.1. Let ϕ be a boolean formula in 3-conjunctive normal form with the variables $X := \{x_1, \ldots, x_n\}$ and the clauses $c_1, \ldots, c_m \subseteq X \times \{+, -\}$.

For every variable x_i, introduce a cycle with $4m$ edges consisting of the vertex set $V_i := \{v_i^j : 1 \leq j \leq 4m\}$ and the edge set

$$E_i := \{e_i^k := \{v_i^k, v_i^{k+1}\} \subseteq V_i\} \cup \{e_i^{4m} := \{v_i^1, v_i^{4m}\}\}.$$

Define the graph $G := (\bigcup_{i=1}^{n} V_i, \bigcup_{i=1}^{n} E_i)$, define the weight function ω by $\omega(e) := 0, e \in E_i$ for any $1 \leq i \leq n$, and define $w_{max} := 1$.
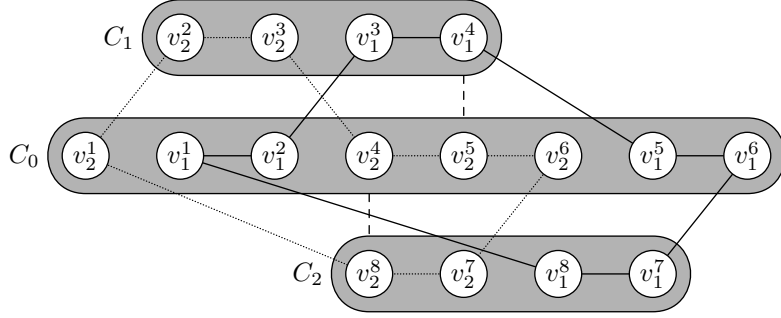
Figure 2.8: Example of Construction 2.3.1 explained in Example 2.3.2.

Inductively define the vertex partition P_m of $V(G)$ and the join set J_m as follows: Let $J_0 = \varnothing$ and let $P_0 := \varnothing$. For every clause c_j introduce the cell

$$C_j := \{v_i^{4j-1} : (x_i, +) \in c_j \vee (x_i, -) \in c_j\} \cup$$
$$\{v_i^{4j-2} : (x_i, +) \in c_j\} \cup$$
$$\{v_i^{4j} : (x_i, -) \in c_j\}.$$

Define $P_i := P_{i-1} \cup \{C_j\}$ and $J_i := J_{i-1} \cup \{\{0, j\}\}$.

Finally, define $C_0 := V(G) \setminus (\bigcup_{j=1}^m C_j)$. The graph G, the weight function ω, the vertex partition $P_m \cup \{C_0\}$ and the join set J_m constitute an instance of CBM.

Example 2.3.2. Figure 2.8 shows an instance of CBM produced from the formula $\phi := (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2)$ by Construction 2.3.1. For simplicity, we chose a formula in 2-conjunctive normal form. The instance comprises the graph G that consists of two directed cycles (solid edges and dotted edges, respectively), three cells C_0, C_1, C_2 forming a partition of $V(G)$ (shaded in gray), and a join set with two joins represented by the dashed lines.

Construction 2.3.1 introduces the solid-edge cycle for variable x_1 and the dotted-edge cycle for variable x_2. The cycle corresponding to x_i has exactly the two perfect matchings

$$M_i^{\text{true}} := \{\{v_i^k, v_i^{k+1}\} : k \text{ odd}\} \text{ and}$$
$$M_i^{\text{false}} := \{\{v_i^k, v_i^{k+1}\} : k \text{ even}\} \cup \{\{v_i^1, v_i^8\}\}.$$

The cell C_1 models the clause $\neg x_1 \vee x_2$ and the vertices are chosen such that only edges of M_1^{false} and edges of M_2^{true} connect the cells C_0 and C_1. Analogously, only edges of M_1^{false} and edges of M_2^{false} connect the cells C_0 and C_2.

There is a correspondence between the clauses a variable x_i satisfies using a particular truth assignment and the joins that are satisfied by matching the cycle that corresponds to x_i using one of the two available matchings. For example, the variable x_1 satisfies both clauses in ϕ when assigned false and no clause when assigned true. Accordingly, the matching M_1^{false} satisfies both the joins $\{0, 1\}$, and $\{0, 2\}$ and the matching M_1^{true} satisfies no join. This holds true analogously for x_2 and thus finding a perfect conjoining matching in G is equivalent to satisfying ϕ.

Lemma 2.3.1. CBM *is NP-hard, even in the unweighted case, even when for every cell C_i in the given vertex-partition of the input graph $G = (V \uplus W, E)$ it holds that $|C_i \cap V| = |C_i \cap W|$ and even when G has maximum degree two.*

Proof. We prove that Construction 2.3.1 is a polynomial-time many-one reduction from 3SAT to CBM. Notice that in instances created by Construction 2.3.1 any matching has weight lower than ω_{max} and, thus, the soundness of the reduction implies that CBM is hard even without the additional weight constraint. Also, since the cells in the instances of CBM are disjoint unions of edges, every cell in the partition P_m contains the same number of vertices from each cell of the graph bipartition.

Concerning Construction 2.3.1, it is easy to check that it is polynomial-time computable. For the correctness we first need the following definition: For every variable $x_i \in X$ let

$$M_i^{\text{true}} := \{e_i^k \in E_i : k \text{ odd}\} \text{ and}$$
$$M_i^{\text{false}} := E_i \setminus M_i^{\text{true}} = \{e_i^k \in E_i : k \text{ even}\}.$$

Observe that all perfect matchings in G are of the form $\bigcup_{i=1}^n M_i^{\nu(i)}$, where ν is some function $\{1, \ldots, n\} \to \{\text{true}, \text{false}\}$. We show that the matching $\bigcup_{i=1}^n M_i^{\nu(i)}$ is a conjoining matching for G with respect to the join set J_m if and only if the truth assignment that assigns each $x_i \in X$ the value $\nu(i)$ is a satisfying truth assignment for ϕ. For this, it suffices to show that for every variable $x_i \in X$ it holds that

$$\{j : (x_i, +) \in c_j\} = \{j : M_i^{\text{true}} \text{ satisfies the join } \{0, j\}\}, \text{ and} \qquad (2.1)$$
$$\{j : (x_i, -) \in c_j\} = \{j : M_i^{\text{false}} \text{ satisfies the join } \{0, j\}\}. \qquad (2.2)$$

We only show that Equation 2.1 holds; Equation 2.2 can be proven analogously. Assume that $(x_i, +) \in c_j$. By Construction 2.3.1 $v_i^{4j-2} \in C_j$, $v_i^{4j-3} \in C_0$ and thus, since

$$\{v_i^{4j-2}, v_i^{4j-3}\} = e^{4j-3} \in M_i^{\text{true}},$$

the matching M_i^{true} satisfies the join $\{0, j\}$. Now assume that $(x_i, +) \notin c_j$, that is, either (1) both $(x_i, \pm) \notin c_j$ or (2) $(x_i, -) \in c_j$. In case (1) we have that V_i and C_j are disjoint and, thus, no matching in $G[V_i]$ can satisfy the join $\{0, j\}$. In case (2) the only edges in E_i that can satisfy the join $\{0, j\}$ are e_i^{4j-2} and e_i^{4j}. However, both these edges are not in M_i^{true} and, thus, this matching cannot satisfy the join $\{0, j\}$. $\qquad \square$

Observation 2.3.1. CBM *is contained in NP and in W[P] when parameterized by the size of the join set.*

Proof. Observe that a minimal matching M that satisfies all joins is a certificate for a yes-instance. Note that M not necessarily has to be perfect. A minimum-weight perfect conjoining matching $M' \supseteq M$, if it exists, can then be found in polynomial time by removing the incident vertices of edges in M from the graph and computing a minimum-weight perfect matching of the remaining vertices. Finding this matching is possible in $\mathrm{O}(mn^2)$ time [12] and it follows that CBM is in NP. Also, generating all minimal matchings that satisfy all joins can be done using a polynomial-time Turing machine using at most $\mathrm{O}(c \log(m))$ nondeterministic steps, where c is the size of the join set: For every join, simply guess an edge that satisfies it. Hence, CBM is in W[P]. $\qquad \square$

Now we can deduce the following theorem:

Theorem 2.3.1. CONJOINING BIPARTITE MATCHING *is NP-complete.*

2.3.1.2 Tractability on Restricted Graphs

In this section we use data reduction rules to show that CBM is fixed-parameter tractable on some restricted classes of input graphs. In particular, we prove that CBM is linear-time decidable on forests (Corollary 2.3.1) and the following theorem:

Theorem 2.3.2. CONJOINING BIPARTITE MATCHING *is solvable in* $O(2^{c(c+1)}n + n^3)$ *time, where c is the size of the join set and when in the bipartite input graph $G = (V_1 \uplus V_2, E)$ each vertex in V_1 has maximum degree two.*

Using this theorem and a reduction from EULERIAN EXTENSION to CBM, we show that EULERIAN EXTENSION is tractable on some restricted instances in Subsection 2.3.2.2. The tractable instances are the preimages of the degree-restricted instances of CBM defined in Theorem 2.3.2.

To prove the Theorem 2.3.2, we use data reduction rules and an observation about matchings in such bipartite graphs as in Theorem 2.3.2. We first give some simple reduction rules and then turn our attention to bipartite graphs with maximum degree two. For these graphs we give a slightly more intricate reduction rule restricting the number of cycles they comprise by some function depending only on the join set size c. These reduced instances are then solved via a search-tree procedure which yields fixed-parameter tractability for CBM on graphs with maximum degree two. A further observation about matchings in bipartite graphs where each vertex in one cell of the bipartition has maximum degree two is then used to generalize the tractability result to Theorem 2.3.2.

In the following, let $G = (V_1 \uplus V_2, E)$ be a bipartite graph, let $\omega : E \to [0, \omega_{max}] \cup \{\infty\}$ be a weight function, let $P = \{C_1, \ldots, C_d\}$ be a vertex partition of G and let J be a join set with respect to G and P.

Simple Data Reduction Rules.

Reduction Rule 2.3.1. *If there is an edge $\{v, w\} \in E$ such that $\deg(v) = 1$, then remove both v and w from G, and remove any join $\{i, j\}$ from J, where $v \in C_i, w \in C_j$. Decrease ω_{max} by $\omega(\{v, w\})$.*

Observation 2.3.2. *Reduction Rule 2.3.1 is correct and can be applied exhaustively in* $O(n + m)$ *time.*

Proof. It is clear that Reduction Rule 2.3.1 is correct because the sought matching is perfect and thus has to match v with w. It can be applied in linear time by first listing all vertices with degree one in linear time and then applying the rule in a depth-first manner outgoing from the degree-one vertices. □

Corollary 2.3.1. CBM *is linear-time solvable on forests.*

Reduction Rule 2.3.2. *If there is a connected component C of G such that $C \subseteq C_j$ for some $1 \leq j \leq c$, then compute a minimum-weight perfect matching M in $G[C]$, remove C from G and decrease ω_{max} by $\omega(M)$.*

Observation 2.3.3. *Reduction Rule 2.3.2 is correct and can be applied exhaustively in* $\mathrm{O}(mn^2)$ *time.*

Proof. The correctness of Reduction Rule 2.3.2 is easy to prove, since for any perfect conjoining matching M' for G we can derive a matching of at most the weight $\omega(M')$ by matching the vertices in $G[C]$ according to M. Hence we can derive a matching of weight at most $\omega(M') - \omega(M)$ in the graph with C removed. Obtaining a matching for G from a matching in the graph G with C removed is trivial.

Applying Reduction Rule 2.3.2 exhaustively can be done by first finding all connected components $D_1, \ldots, D_k$ that are contained in one cell in linear time and then computing a minimum-weight perfect matching in the graph $G[\bigcup_{i=1}^{k} D_i]$ in $\mathrm{O}(mn^2)$ time [12]. Then, deleting the affected vertices is possible in linear time. $\qquad\square$

Reduction Rule for Maximum Degree Two. Now let $G = (V_1 \uplus V_2, E)$ be a bipartite graph with maximum degree two of an instance of CBM that is preprocessed with Reduction Rule 2.3.1 and Reduction Rule 2.3.2. In this graph, any degree-one vertices have been deleted and thus each vertex has degree two. It follows that G consists of connected components each of which is a cycle of even length—because G is bipartite. Thus every connected component has exactly two perfect matchings. To describe a third reduction rule, we need the following definitions:

Definition 2.3.3. For every connected component, that is, every cycle D contained in G, denote by $M_1(D)$ a minimum-weight perfect matching of D with respect to ω and denote by $M_2(D) := E(D) \setminus M_1(D)$, that is, the other perfect matching of D. Furthermore, define

$$\sigma_1(D) := \{j \in J : \exists e \in M_1(D) : e \text{ satisfies } j\},$$
$$\sigma_2(D) := \{j \in J : \exists e \in M_2(D) : e \text{ satisfies } j\}$$

and the *signature* $\sigma(D)$ of D as $(\sigma_1(D), \sigma_2(D))$. We say that two signatures $\sigma(A), \sigma(B)$ are *equal* and we write $\sigma(A) \equiv \sigma(B)$, if

$$(\sigma_1(A) = \sigma_1(B) \wedge \sigma_2(A) = \sigma_2(B)) \vee$$
$$(\sigma_1(A) = \sigma_2(B) \wedge \sigma_2(A) = \sigma_1(B)).$$

Reduction Rule 2.3.3. *Let $S = \{D_1, \ldots, D_j\}$ be a maximal set of connected components of G such that $\sigma(D_1) \equiv \ldots \equiv \sigma(D_j)$ and $j \geq 2$. Let $M_1^* = \bigcup_{k=1}^{j} M_1(D_k)$, let $D_l \in S$ such that $\omega(M_2(D_l)) - \omega(M_1(D_l))$ is minimum and let $M_1^{\sim} = M_1^* \setminus M_1(D_l)$.*

(i) If the matching M_1^* is conjoining for the join set $\sigma_1(D_1) \cup \sigma_2(D_1)$, then remove each component in S from G, remove each join in $\sigma_1(D_1) \cup \sigma_2(D_1)$ from the join set J, and reduce ω_{max} by $\omega(M_1^*)$.

(ii) If the matching M_1^* is not conjoining for the join set $\sigma_1(D_1) \cup \sigma_2(D_1)$ remove each component in $S \setminus \{D_l\}$ from G, remove any join in $\sigma_1(D_1)$ from the join set J, and reduce ω_{max} by $\omega(M_1^{\sim})$.

In either case, update the partition P accordingly.

Lemma 2.3.2. *Reduction Rule 2.3.3 is correct.*

Proof. Let $G = (V_1 \uplus V_2, E)$ be a graph with maximum degree two, let $\omega : E \to [0, \omega_{max}] \cup \{\infty\}$ be a weight function, let $P = \{C_1, \ldots, C_c\}$ be a vertex partition of G and let J be a join set with respect to G and P. The objects $G, \omega, \omega_{max}, P$, and J constitute an instance I of CBM. Furthermore, let the graph G', the weight function ω, the maximum weight ω'_{max}, the vertex partition P', and the join set J' with respect to G' and P' constitute the instance I' that is obtained from I by applying Reduction Rule 2.3.3 with the set $S = \{D_1, \ldots, D_j\}$ as defined there.

Let M be a perfect J-conjoining matching for G with $\omega(M) \leq \omega_{max}$ and assume that the matching $M_1^* = \bigcup_{k=1}^{j} M_1(D_k)$ is conjoining for the join set $\sigma_1(D_1) \cup \sigma_2(D_1)$. Then either $M_1^* \subseteq M$, or we can obtain another perfect J-conjoining matching with weight at most $\omega(M)$ that satisfies this property. Without loss of generality assume that $M_1^* \subseteq M$. Then $M \setminus M_1^*$ is a perfect J'-conjoining matching for G' with weight $\omega(M) - \omega(M_1^*) \leq \omega'_{max}$.

Now assume that M_1^* is not conjoining for the join set $\sigma_1(D_1) \cup \sigma_2(D_1)$. Then either

(1) $M_1^* \subseteq M$ or

(2) there is an integer n such that $M_2(D_n) \subseteq M$.

We first show that, in case (2), we may assume without loss of generality that n is unique and that $n = l$ as in Reduction Rule 2.3.3. Otherwise we can find another perfect J-conjoining matching with weight at most $\omega(M)$ that satisfies this property: Since M_1^* is not conjoining for the join set $\sigma_1(D_1) \cup \sigma_2(D_1)$, it holds that

$$\sigma_1(D_1) = \ldots = \sigma_1(D_j), \qquad \text{and} \qquad \sigma_2(D_1) = \ldots = \sigma_2(D_j),$$

because all signatures of the components in S are equal by prerequisite of Reduction Rule 2.3.3. If n is not unique, there are n, m such that $M_2(D_n), M_2(D_m) \subseteq M$. However, by definition $\omega(M_1(A)) \leq \omega(M_2(A))$ and if we substitute $M_1(D_m)$ for $M_2(D_m)$ in M, the resulting matching has at most the same weight and is still J-conjoining because $\sigma_2(D_n) = \sigma_2(D_m)$. Hence we can assume that n is unique. We can also assume that $n = l$ because by definition of l

$$\omega(M_2(D_l)) - \omega(M_1(D_l)) \leq \omega(M_2(D_n)) - \omega(M_1(D_n))$$

and thus we can substitute $M_1(D_n)$ for $M_2(D_n)$ and $M_2(D_l)$ for $M_1(D_l)$ in the matching M to obtain a perfect J-conjoining matching of at most the same weight. Consider the matching $M_1^\sim = \bigcup_{1 \leq k \leq j, k \neq l} M_1(D_k)$. Both in case (1) and in case (2), when assuming that $n = l$ is unique, $M \setminus M_1^\sim$ is a perfect J'-conjoining matching for G' of weight $\omega(M) - \omega(M_1^\sim) \leq \omega'_{max}$.

We now have that if I is a yes instance then I' is a yes instance. For the other way round, assume that M' is a perfect J'-conjoining matching for G' of weight $\omega(M') \leq \omega'_{max}$. Assume that each component in S of G has been removed in G' by Reduction Rule 2.3.3. Then the matching $M' \cup M_1^*$ for G is perfect, J-conjoining and of weight $\omega(M) + \omega(M_1^*) \leq \omega_{max}$. Now assume only the component D_l of the components in S is still present in G'. Then, the matching $M \cup M_1^\sim$ is a perfect J-conjoining matching for G of weight $\omega(M) + \omega(M_1^\sim) \leq \omega_{max}$. $\square$

Lemma 2.3.3. *Reduction Rule 2.3.3 can be applied exhaustively in* $\mathrm{O}(n^3)$ *time.*

Proof. To apply Reduction Rule 2.3.3 once, we can first search for a set of components S as defined there by first finding all connected components in linear time. Then we find out the signature of each connected component. For this, we first compute a minimum-weight perfect matching for every connected component in overall $O(m)$ time by simply iterating over the edges in each component, alternatingly summing up the edge weights and choosing the lower one of the two values. We annotate every edge with whether it is contained in the minimum-weight matching or not and which join it satisfies, if any, in $O(m^2)$ time. We then iterate over every edge and add the information saved in the annotation to the signature of the connected component it is contained in.

Having computed the signatures, we create a map in $O(n \log(n))$ time that maps every signature present to the list of connected components that have this signature. We then simply iterate over every list present in the map to obtain a maximal list of components that have the same signature or decide that there is no such list with at least two elements. This is possible in $O(n)$ time.

The removal of the connected components and joins, the update of ω_{max} and the partition P is then possible in linear time, because the matchings for each component have already been computed and thus the overall running time is $O(m^2 + n \log n)$. Observe that in graphs with exactly degree two $m \in O(n)$ and thus we can derive a running time bound in $O(n^2)$.

In any application either no set S is found and thus the procedure terminates, or at least 4 vertices are deleted—this is the minimum size of a connected component. Hence the procedure can be applied at most n times and exhaustively applying Reduction Rule 2.3.3 takes $O(n^3)$ time. $\qquad\square$

Observation 2.3.4. *When Reduction Rule 2.3.3 cannot be applied anymore, the input graph contains at most 2^{c+1} components, where c is the size of the join set.*

Proof. When there are c joins in a join set, then there are at most 2^{c+1} signatures. For each signature, there is at most one connected component when Reduction Rule 2.3.3 is not applicable. $\qquad\square$

Lemma 2.3.4. *CBM is solvable in $O(2^{c(c+1)}n + n^3)$ time on graphs with maximum degree two, where c is the size of the join set.*

Proof. This follows from exhaustively applying Reduction Rule 2.3.1, Reduction Rule 2.3.2, and Reduction Rule 2.3.3 and then invoking a search tree algorithm. The algorithm chooses one join, branches into choosing any component that contains an edge that satisfies the join, matches the component accordingly and then recurses until every join is satisfied. Since there are at most 2^{c+1} components in the preprocessed graph, every branching-step invokes at most 2^{c+1} recursive calls. The recursion depth is obviously at most c. In every call at most $O(n)$ time is spent finding components satisfying the chosen join and thus we can derive a running time bound of $O((2^{c+1})^c n) = O(2^{c(c+1)}n)$ for the search tree algorithm. The preprocessing rules take $O(n^3)$ time by Observation 2.3.2, Observation 2.3.3, Lemma 2.3.3, and by the fact that $m \in O(n)$ in graphs with degree at most two. Thus the overall running time bound is $O(2^{c(c+1)}n + n^3)$. $\qquad\square$

Perfect Matchings in Graphs with Maximum Degree Two.   Now let $G = (V_1 \uplus V_2, E)$ be a bipartite graph where each vertex in V_1 has maximum degree

two. We show that if G has a perfect matching, it will be preprocessed by Reduction Rule 2.3.1 such that each vertex has degree exactly two.

Lemma 2.3.5. *If G has a perfect matching, every connected component of G contains at most one cycle as subgraph.*

Proof. We show that if G contains a connected component that contains two cycles c_1, c_2 as subgraphs, then G does not have a perfect matching. First assume that c_1, c_2 are vertex-disjoint. Then, there is a path p from a vertex $v \in V(c_1)$ to a vertex $w \in V(c_2)$ such that p is vertex-disjoint from c_1 and c_2 except for v, w. It is clear that both $v, w \in V_2$ because they have degree three. Consider the vertices $V_1^{cp} := (V(c_1) \cup V(p) \cup V(c_2)) \cap V_1$ and the set $V_2^{cp} := (V(c_1) \cup V(p) \cup V(c_2)) \cap V_2$. The set V_2^{cp} is the set of neighbors of vertices in V_1^{cp}, because they have degree two and thus have neighbors only within p, c_1, and c_2. It is $|V_1^{cp}| = (|E(c_1)| + |E(p)| + |E(c_2)|)/2$ since neither of these paths and cycles overlap in a vertex in V_1. However, it is $|V_2^{cp}| = |V_1^{cp}| - 1$ because c_1 and p overlap in v and c_2 and p overlap in w. This is a violation of Hall's condition—recall the definition of Hall's condition in Theorem 1.1.2—and thus G does not have a perfect matching.

The case where c_1 and c_2 share vertices can be proven analogously. (Observe that then there is a subpath of c_2 that is vertex-disjoint from c_1 and contains an even number of edges.) $\qquad\square$

Proof of Theorem 2.3.2. Consider applying Reduction Rule 2.3.1 to a graph $G = (V_1 \uplus V_2, E)$ such that each vertex in V_1 has maximum degree two and such that G has a perfect matching. This has to yield a graph that is a collection of vertex-disjoint cycles because in every connected component there is at most one cycle as subgraph (Lemma 2.3.5). Hence, every component consists of a cycle with a collection of pairwise vertex-disjoint paths incident to it. These paths are completely reduced by Reduction Rule 2.3.1 and all that remains is either the cycle or nothing. Thus, in order to cope with graphs G as above, we can modify the algorithm from Lemma 2.3.4: If the application of Reduction Rule 2.3.1 does not yield a graph that is a collection of vertex-disjoint cycles, we can abort the procedure because it cannot yield a perfect matching. This can be checked in linear time and thus, Theorem 2.3.2 now directly follows. (Notice that the running time bound of Lemma 2.3.4 does not increase, since in graphs G as above that have a perfect matching it also holds that $m \in \mathrm{O}(n)$.) $\qquad\square$

2.3.2 The Relationship between Eulerian Extension and Matching

In this section we show that CBM parameterized by the size of the join set and EE parameterized by the number of connected components in the input graph are parameterized equivalent. To this end, we first give a reduction from EECA to CBM. This reduction also yields an efficient algorithm for a restricted variant of EE. Second, we give a reduction from CBM to EEA. The equivalence of EE and CBM then follows from the reductions given in Lemma 2.2.4 and Theorem 2.2.2 in Subsection 2.2.2.

2.3.2.1 Reducing EECA to CBM

We first reduce EECA to CBM. In order to simplify our reduction, we reduce from EE∅CA instead (see page 30 in Subsection 2.2.2). We know that EE∅CA and EECA are equivalent from Observation 2.2.3.

As we have observed in Observation 2.1.1 we have to draw paths between unbalanced vertices in order to make them balanced and to ultimately make the input graph Eulerian. These paths also have to connect all components of the input graph. The basic structure of these paths is made explicit by the advice in EE∅CA and thus we do not have to concern ourselves with finding a suitable order of components for these paths. We simply have to realize every hint to connect the graph and then balance all remaining vertices.

Reduction Outline. The basic ideas of our reduction are to use vertices of positive balance and negative balance in an instance of EE∅CA as the two cells of the graph bipartition in a designated instance of CBM. Edges between vertices in the new instances represent shortest paths between them that consist of allowed extension arcs in the original instance. Every connected component in the original instance is represented by a cell in the vertex partition in the matching instance and hints are basically modeled by joins.

We proceed with an intuitive description of the reduction and then go into the details in Construction 2.3.2. The construction is then followed by a correctness proof. For the descriptions, we first need the following definition.

Definition 2.3.4. Let G be a directed multigraph with the connected components $V_1, \ldots, V_c$ and let H be a cycle-free advice for G. For every hint $h \in H$ we define connect$(h) = \{i, j\}$, where C_i, C_j are the components corresponding to the initial and terminal vertices of h.

Intuitive Description. First, consider an instance $I_{\text{EE}\emptyset\text{CA}}$ of EE∅CA that consists of the graph G, the weight function $\omega : V \times V \to [1, \omega_{max}] \cup \{\infty\}$ and a cycle-free minimal connecting advice H that contains only hints of length one. We will deal with longer hints later. We create an instance I_{CBM} of CBM by first defining $B_0 = (I_G^+ \uplus I_G^-, E_0)$ as a bipartite graph. Here, the set E_0 consists of all edges $\{u, v\}$ such that $u \in I_G^+$, $v \in I_G^-$, and $\omega(u, v) < \infty$. This serves the purpose of modeling the structure of allowed arcs in the matching instance—we come back to this in Subsection 2.3.2.2. Second, we derive a vertex partition $\{V_1', \ldots, V_c'\}$ of B_0 by intersecting the connected components of G with $(I_G^+ \uplus I_G^-)$. The vertex-partition obviously models the connected components in the input graph, and the need of connecting them according to the advice H is modeled by an appropriate join-set J_0, defined as $\{$connect$(h) : h \in H\}$. Finally, we make sure that matchings also correspond to Eulerian extensions weight-wise, by defining the weight function $\omega'(\{u, v\})$ for every $u \in I_G^+, v \in I_G^-$ as $\omega(u, v)$ with $\omega'_{max} = \omega_{max}$.

By Observation 2.1.4 we may assume that every hint in H of length one is realized by a single arc. Since the advice connects all connected components, by the same observation, we may assume that all other trails in a valid Eulerian extension have length one (Observation 2.1.4 also holds for the connected graph obtained by adding the realizations of all hints to the input graph). Finally, by Lemma 2.1.1, we may assume that every vertex has at most one incident incoming

(a) EE∅A instance

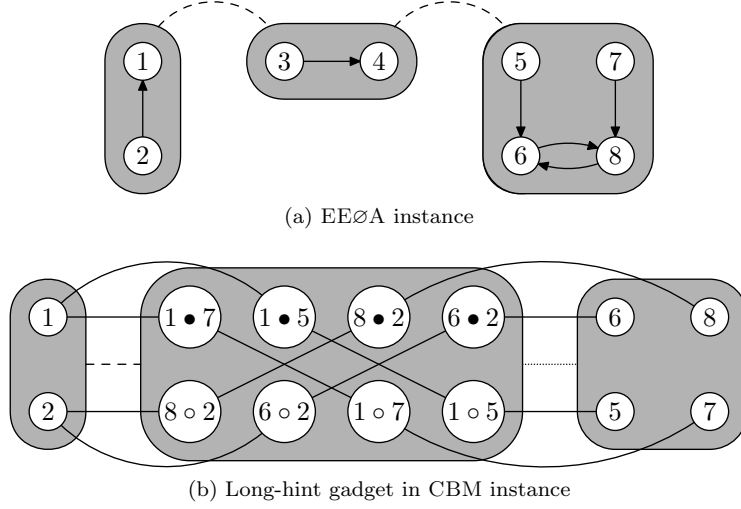(b) Long-hint gadget in CBM instance

Figure 2.9: Example for the long-hint gadget used in Construction 2.3.2, explained in the corresponding intuitive description.

or outgoing arc in the extension and, hence, we get an intuitive correspondence between the matchings and Eulerian extensions.

To model hints of length at least two, we utilize gadgets similar to the one shown in Figure 2.9. On the top, an instance $I_{\text{EE∅A}}$ is shown, consisting of a graph with three connected components and an advice that contains a single hint h (dashed lines). Below in Figure 2.9b a part of an instance of CBM is shown, which comprises the cells that correspond to the initial and terminal vertices of h and a gadget to model h. The gadget consists of some new vertices which are put into a new cell which is connected by two joins (dashed and dotted lines) to the cells corresponding to the initial and terminal vertices of h.

The gadget comprises two vertices ($u \circ v$ and $u \bullet v$) for every pair (u, v) of vertices with one vertex in the component the hint starts and one in the component the hint ends. The vertices $u \circ v$ and $u \bullet v$ are adjacent and each of these two vertices is connected with one vertex of the pair it represents. The edge $\{u \bullet v, u\}$ is weighted with the cost it takes to connect u, v with a path p such that $\mathbb{C}_G(p) = h$ that is, a path that realizes h. The other edges have weight 0. Intuitively these three edges in the gadget represent one concrete realization of h. If $u \circ v$ and $u \bullet v$ are matched, this means that this specific path does not occur in a designated Eulerian extension. However, by adding the vertices of the gadget as cell to the vertex partition and by extending the join set to the gadget, we enforce that there is at least one outgoing edge that is matched. If $v \bullet u$ is matched with v, then $v \circ u$ must be matched with u and vice versa, otherwise the matching could not be perfect. This introduces an edge to the matching that has weight corresponding to a path that realizes h.

Construction 2.3.2. Let the directed multigraph $G = (V, A)$, the weight function $\omega : V \times V \to [1, \omega_{max}] \cup \{\infty\}$ and the advice H constitute an instance of EE∅CA. Let $V_1, \dots, V_c$ be the connected components of G.

Let $H^{=1}$ be the set of hints of length one in H and let $H^{\geq 2}$ be the set of hints

in H that have length at least two. Define J_0 by the set $\{\text{connect}(h) : h \in H^{=1}\}$. Let $W_0^1 := I_G^+$, $W_0^2 := I_G^-$, and let $B_0 = (W_0^1 \uplus W_0^2, E_0)$ be a bipartite graph where
$$E_0 := \{\{u, v\} : u \in I_G^+ \wedge v \in I_G^- \wedge \omega(u, v) < \infty\}.$$

Define $V_i' := V_i \cap (I_G^+ \cup I_G^-)$, $1 \leq i \leq c$, and $\omega_0'(\{u, v\}) := \omega(u, v)$ where $\{u, v\} \in E$, $u \in I_G^+$.

Let $h_1^{\geq 2}, \ldots, h_j^{\geq 2}$ be the hints in $H^{\geq 2}$. Inductively define B_k, V_{c+k}', ω_k' and J_k, $1 \leq k \leq j$, as follows: Let $\text{connect}(h_k^{\geq 2}) = \{o, p\}$. Introduce the vertex sets

$$U_1 := \{v \circ u : v \in I_G^+ \cap V_o \wedge u \in I_G^- \cap V_p \wedge \omega(\text{minpath}(G, \omega, h_k^{\geq 2}, v, u)) < \infty\} \cup$$
$$\{v \circ u : v \in I_G^- \cap V_o \wedge u \in I_G^+ \cap V_p \wedge \omega(\text{minpath}(G, \omega, h_k^{\geq 2}, u, v)) < \infty\},$$

and $U_2 := \{v \bullet u : v \circ u \in U_1\}$. Introduce the edge sets

$$E_k^1 := \{\{v \circ u, v\} : v \in I_G^- \wedge v \circ u \in U_1\},$$
$$E_k^2 := \{\{v \bullet u, v\} : v \in I_G^+ \wedge v \bullet u \in U_2\}, \text{ and}$$
$$E_k^3 := \{\{v \circ u, v \bullet u\} : v \circ u \in U_1 \wedge v \bullet u \in U_2\}.$$

Set $E_k := E_k^1 \cup E_k^2 \cup E_k^3$, and set the graph

$$B_k := ((W_{k-1}^1 \cup U_1) \uplus (W_{k-1}^2 \cup U_2), E_{k-1} \cup E_k),$$

set $V_{c+k}' := U_1 \cup U_2$, set $J_k := J_{k-1} \cup \{\{o, c+k\}, \{p, c+k\}\}$ and the weight-function as follows:

$$\omega_k'(\{u, v\}) := \begin{cases} \omega_{k-1}'(\{u, v\}), & \{u, v\} \in E_{k-1} \\ 0, & \{u, v\} \in E_k^1 \cup E_k^3 \\ \omega(\text{minpath}(G, \omega, h_k^{\geq 2}, v, w)), & \{u, v\} = \{v \bullet w, v\} \in E_k^2 \end{cases}$$

Then the graph B_j, the weight function ω_j', the vertex partition $P := \{V_1, ..., V_{c+j}\}$ and the join set C_j constitute an instance of CBM.

For the remainder of this section, let the directed multigraph $G = (V, A)$, the weight function $\omega : V \times V \to [1, \omega_{max}] \cup \{\infty\}$ and the cycle-free minimal connecting advice H constitute an instance of EE$\varnothing$CA and let the bipartite graph $B := B_j$, the weight function $\omega' := \omega_j'$ with the maximum weight ω_{max}, the vertex partition P and the join set $J := J_j$ as in Construction 2.3.2 constitute an instance of CBM.

Lemma 2.3.6. *Let E be an Eulerian extension for G that heeds the advice H. Then there is a perfect conjoining matching M for B with $\omega'(M) \leq \omega(E)$.*

Proof. We construct the matching successively by first looking at every long-path gadget in B and then matching the remaining vertices.

Consider the cell $V_{c+k}' \in P$ for $k > 0$. There are two joins $\{c+k, o\}$ and $\{c+k, p\}$ in J. Thus, there is a path hint h from V_o to V_p in H. This means that, there is a path s in E that starts in a vertex $v \in V$ in the component V_o and ends in a vertex $u \in V$ in V_p. The weight $\omega(s)$ is at least $\omega(\text{minpath}(G, \omega, h, u, v))$ (Observation 2.2.1). Thus we may match $u \bullet v$ with v, $u \circ v$ with u (this costs

51

weight $\omega(\mathrm{minpath}(G, \omega, h, u, v)))$ and every other pair $w \bullet x$ and $w \circ x$ in V'_{c+k} with each other (this costs weight 0). Matching like this, we obtain a matching for the long-hint gadget of h that fulfills its two joins and is perfect. The weight of the matching is at most the realization of h in E.

Because of shortest-path preprocessing (Transformation 2.1.2) and Observation 2.1.5 we may assume that there is a set of paths in E that is edge-disjoint and realizes all hints in H (otherwise we may obtain an Eulerian extension of at most the same weight that has this property). Because of this, we may find a matching $M^{\geq 2}$ for B that satisfies the joins of every long-hint gadget and is perfect with respect to the vertex set of each long-hint gadget—as in the previous paragraph, iterated for every gadget. Furthermore, $\omega'(M^{\geq 2})$ is lower than the weight of all paths in E that realize hints of length at least two in H.

Now it is easy to extend $M^{\geq 2}$ to a conjoining matching $M^{\geq 1}$ for B and J just by adding matching edges between vertices that realize hints of length one in E. We may assume by Observation 2.1.4 that each hint of length one is realized by a single arc in E. The weight of matching edges is exactly the cost of the direct arc between the corresponding vertices. Because of this, we maintain that $\omega'(M^{\geq 1})$ is at most the weight of all paths in E that realize hints.

Finally, we have to extend $M^{\geq 1}$ to a perfect matching M by matching the remaining non-gadget vertices. We can do this by looking at paths in E that start and end in the vertices in G, corresponding to still unmatched vertices in B. A set of such paths must exist, because each such vertex has at least one incident arc in E and because, by Observation 2.1.1, maximal-length open trails in Eulerian extensions start and end in unbalanced vertices. The edges between initial and terminal vertices of those paths in B have at most the weight of such a path (because of shortest-path preprocessing and because they have weight corresponding to the direct arc). Thus, we can add those edges to $M^{\geq 1}$, obtaining an edge set M. This set is a matching for B that is perfect, conjoining and $\omega'(M) \leq \omega(E)$. $\qquad\square$

Lemma 2.3.7. *Let M be a perfect conjoining matching for B. We can construct an Eulerian extension E for G that heeds the advice H such that $\omega(E) = \omega'(M)$.*

Proof. We simply look at every matching edge that has non-zero weight and add a corresponding path to a designated Eulerian extension E of G: For non-gadget matching edges (edges that match vertices in $V'_1, \ldots, V'_c$) the corresponding path is the direct arc between the two vertices in G. For edges that match a vertex v in a cell V'_o, $1 \leq o \leq c$ and a vertex $u \bullet v \in V'_{c+k}$, $1 \leq k \leq j$, where $u \in V'_p$, $1 \leq p \leq c$, the corresponding path is $\mathrm{minpath}(G, \omega, h_k, u, v)$. Here, h_k is the path in H that lead to the introduction of V'_{c+k} in Construction 2.3.2.

We immediately see that $\omega(E) = \omega'(M)$. Also, it is clear that every hint of length one in H is realized in E because every hint h^1 of length one leads to the pair $\mathrm{connect}(p^1)$ in J. Hints $p^{\geq 2}$ of length two are also realized, because every such path leads to a cell V'_{c+k}, $1 \leq k \leq j$ and also leads to the corresponding joins $\{o, c+k\}$ and $\{p, c+k\}$ in J, where $\{o, p\} = \mathrm{connect}(h^{\geq 2})$. Thus, E heeds the advice H. Since M is a perfect matching, every unbalanced vertex in G is the initial or terminal vertex of exactly one path added to E in the above paragraph. By Lemma 2.1.1 we may assume that this suffices to make every vertex in $G + E$ balanced. Also, $G + E$ is connected, because E heeds the advice H. $\qquad\square$

Lemma 2.3.8. *Construction 2.3.2 is computable in $\mathrm{O}(|H|n^4 + m)$ time.*

Proof. Computing B_0 takes $O(n^2)$ time. To compute J_0 one needs $O(|H|)$ time by iterating over every path in H. Computing the initial partition $\{V_1', \ldots, V_c'\}$ takes $O(n + m)$ time and the initial weight function ω_0' can also be computed within this time. Hence, creating the initial instance is possible in $O(n^2 + m)$ time.

Regarding adding the gadget for one path in H, to compute the sets U_1 and U_2, $O(n^4)$ time is suffices, because n^2 instances of minpath have to be computed, each taking $O(n^2)$ time (Lemma 2.2.1). There are only three edges in the gadget for every vertex $v \in U_1$, thus computing the edge sets does not increase the running time bound. For the weight function we can reuse the values of minpath computed for every pair of vertices $v \in I_G^+, u \in I_G^-$ and thus we can conclude an overall running time bound of $O(|H|n^4 + m)$. $\qquad\square$

Now the following theorem follows:

Theorem 2.3.3. Eulerian Extension with Minimal Connecting Advice *is polynomial-time many-one reducible to* Conjoining Bipartite Matching. *The corresponding reduction function is a parameterized reduction with respect to the parameters number of components in the graph of* EECA *and join set size in* CBM.

Proof. By Observation 2.2.3, there is a polynomial-time many-one reduction from EECA to EE∅CA. This reduction at most decreases the number of components in the input graph. By Lemma 2.3.6 and Lemma 2.3.7 there is a many-one reduction from EE∅CA to CBM . Since the construction is polynomial-time computable (Lemma 2.3.8), since for every hint in the advice there are at most two joins, and since the number of hints is bounded by the number of components in the input graph to EECA (Observation 2.2.2) it follows that Construction 2.3.2 is a parameterized polynomial-time many-one reduction. $\quad\square$

Corollary 2.3.2. Eulerian Extension *is parameterized Turing reducible to* Conjoining Bipartite Matching *with respect to the parameters number of components in input graph and join set size.*

Proof. The statement follows from Lemma 2.2.4 and Theorem 2.3.3. $\qquad\square$

2.3.2.2 Islands of Tractability for EE

Using the reduction given in Construction 2.3.2, we can gather the fruit of our work in Subsection 2.3.1 where we showed restricted fixed-parameter tractability of CBM with respect to the join set size.

Corollary 2.3.3. *Let the graph G and the weight function ω constitute an instance I_{EE} of* EE. *Let c be the number of connected components in G. Furthermore,*

(i) let the set A_A of allowed arcs with respect to ω not contain a path or cycle of length at least two,

(ii) let the underlying graph of the directed graph $(V, V \times V)\langle A_A \rangle$ be a forest, and

(iii) let G contain only vertices with balance between -1 and 1.

Then, it is decidable in $O(16^{c\log(c)}(cn^4 + m))$ time whether I_{EE} is a yes-instance.

Proof. Observe that such instances are invariant under Transformation 2.1.1 and Transformation 2.1.2. Thus, we may directly apply the reduction from EE to EECA given in Lemma 2.2.4 that runs in time $O(16^{c \log(c)}(c + n + m))$. Also, there is no valid advice that contains hints of length two for such graphs. Thus, we can apply Construction 2.3.2—running in $O(cn^4 + m)$ time by Lemma 2.3.8— to map the instances of EECA to instances of CBM that comprise bipartite graphs that are forests. By Corollary 2.3.1, these instances are solvable in linear time. $\square$

Corollary 2.3.4. *Let the graph G and the weight function ω constitute an instance I_{EE} of* EE. *Let c be the number of connected components in G. Furthermore,*

(i) let the set of allowed arcs with respect to ω not contain a path or cycle of length at least two,

(ii) let G contain only vertices with balance between -1 and 1,

(iii) let every vertex in I_G^+ (every vertex in I_G^-) have only outgoing allowed arcs (incoming allowed arcs),

(iv) for every connected component C of G, let either all vertices in $I_G^+ \cap C$ have at most two incident allowed arcs or let all vertices in $I_G^- \cap C$ have at most two incident allowed arcs.

Then, it is decidable in $O(2^{c(c+\log(2c^4))}(n^4 + m))$ time whether I_{EE} is a yes-instance.

Proof. The proof is analogous to Corollary 2.3.3 by substituting the algorithm we gave in Theorem 2.3.2 for Corollary 2.3.1. This leads to a running time bound of $O(16^{c \log(c)}(cn^4 + m + 2^{c(c+1)}n)) \subseteq O(2^{c(c+\log(2c^4))}(n^4 + m))$. $\square$

2.3.2.3 Reducing CBM to EEA

To reduce CBM to EEA we first observe that for every instance of CBM there is an equivalent instance such that every cell in the input vertex-partition contains equal numbers of vertices from both cells of the graph bipartition. This observation enables us to model cells as connected components and vertices in the bipartite graph as unbalanced vertices in the designated instance of EEA.

We first need the following auxiliary observations:

Observation 2.3.5. *Let $G = (V_1 \uplus V_2, E)$ be a bipartite graph such that $|V_1| = |V_2|$ and let the set $P = \{C_1, \ldots, C_k\}$ be a partition of the vertices in G. It holds that*

$$\sum_{i:|C_i \cap V_1| > |C_i \cap V_2|} |C_i \cap V_1| - |C_i \cap V_2| = \sum_{i:|C_i \cap V_1| < |C_i \cap V_2|} |C_i \cap V_2| - |C_i \cap V_1|.$$

Proof. Observe that the equation holds if and only if $|V_1| = |V_2|$: Without loss of generality we may assume that there are no cells C_i with $|C_i \cap V_1| = |C_i \cap V_2|$ because these do contribute summands to the equation. Then we can transpose the equation such that the left-hand side reads as follows

$$\sum_{i:|C_i \cap V_1| > |C_i \cap V_2|} |C_i \cap V_1| + \sum_{i:|C_i \cap V_1| < |C_i \cap V_2|} |C_i \cap V_1|.$$

This is equal to $|V_1|$. Analogously, the left-hand side in the transposed formula is equal to $|V_2|$. $\square$

Lemma 2.3.9. *For every instance of* CBM *there is an equivalent instance comprising the bipartite graph* $G = (V_1 \uplus V_2, E)$*, the vertex partition* $P = \{C_1, \ldots, C_{k+1}\}$ *and the join set* J*, such that*

(i) for every $1 \leq i \leq k + 1$ *it holds that* $|V_1 \cap C_i| = |V_2 \cap C_i|$*, and*

(ii) the graph $(P, \{\{C_i, C_j\} : \{i, j\} \in J\})$ *is connected.*

Furthermore, this equivalent instance contains at most one cell more than the original instance.

Proof. We first prove that there is an equivalent instance corresponding to statement (i) and then turn to statement (ii). Let the bipartite graph $G = (V_1 \uplus V_2, E)$, the weight function $\omega : E \to [0, \omega_{max}] \cup \{\infty\}$, the vertex partition $P = \{C_1, \ldots, C_k\}$ and the join set J constitute an instance I_{CBM} of CBM. First observe that if I_{CBM} is a yes-instance then $|V_1| = |V_2|$, otherwise there could not be a perfect matching. Thus, if $|V_1| \neq |V_2|$ we may simply output a trivial no-instance for which the statement of the lemma holds. Otherwise, by Observation 2.3.5, the following procedure can be carried out: Add a new cell C_{k+1} to P with

$$\sum_{i : |C_i \cap V_1| > |C_i \cap V_2|} |C_i \cap V_1| - |C_i \cap V_2|$$

vertices in V_1 and the same number of vertices in V_2, and modify the graph G and each cell $C_i \in P$ with $\alpha := |C_i \cap V_1| - |C_i \cap V_2| > 0$ as follows: Add the new vertices $v_1, \ldots, v_\alpha$ to V_2 and to the cell C_i, and add an edge from v_j to a vertex in $C_{k+1} \cap V_1$ for every $1 \leq j \leq \alpha$ and such that every vertex in C_{k+1} gets at most one incident edge. Proceed analogously for cells C_i with $\alpha := |C_i \cap V_2| - |C_i \cap V_1| > 0$ by adding vertices to V_1 and adding corresponding edges to C_{k+1}. Finally, expand the weight function ω to the new edges by giving each of them weight 0.

This construction is obviously correct, since each new vertex can only be matched to its corresponding vertex in C_{k+1}.

Concerning statement (ii), assume that the statement does not hold for a instance that contains the vertex partition $P = \{C_1, \ldots, C_k\}$ and a join set J. We greedily choose two cells C_i, C_j that are in different connected components in the "cell-join graph" $(P, \{\{C_i, C_j\} : \{i, j\} \in J\})$, remove them from P, add the cell $C_k := C_i \cup C_j$ and update J accordingly—that is, we replace every join $\{m, l\} \in J$ where $m \in \{i, j\}$ by the join $\{k, l\}$. This is correct because all joins satisfied by any solution M for the new instance are also satisfied by M in the original instance and vice versa. Iterating the merging of cells in differ ent connected components makes the cell-join graph connected and the statement follows. $\square$

Description of the Reduction. To reduce instances of CBM that conform to statement (i) and (ii) of Lemma 2.3.9 to instances of EEA we use the simple idea of modelling every cell as connected component, vertices in V_1 as vertices with balance -1, vertices in V_2 as vertices with balance 1, and joins as hints.

Construction 2.3.3. Let the bipartite graph $B = (V_1 \uplus V_2, E)$, the weight function $\omega : E \to [0, \omega_{max}] \cup \{\infty\}$, the vertex partition $P = \{C_1, \ldots, C_k\}$ and the join set J constitute an instance I_{CBM} of CBM such that I_{CBM} corresponds to Lemma 2.3.9(i) and (ii).

Let $v_1^1, v_1^2, \ldots, v_{n/2}^1, v_{n/2}^2$ be a sequence of all vertices chosen alternatingly from V_1 and V_2. Define the graph $G = (V, A) := (V_1 \cup V_2, A_1 \cup A_2)$ where the arc sets A_1 and A_2 are defined as follows: $A_1 := \{(v_i^1, v_i^2) : 1 \leq i \leq n/2\}$. For every $1 \leq j \leq k$ let $C_j = \{v_1, \ldots, v_{j_k}\}$, set

$$A_2^j := \{(v_i, v_{i+1}) : 1 \leq i \leq j_k - 1\} \cup \{(v_{j_k}, v_1\}$$

and define $A_2 := \bigcup_{j=1}^k A_2^j$. Define a new weight function ω' for every pair of vertices $(u, v) \in V \times V$ by

$$\omega'(u, v) := \begin{cases} \omega(\{u, v\}), & u \in V_2, v \in V_1, \{u, v\} \in E \\ \infty, & \text{otherwise.} \end{cases}$$

Finally, derive an advice H for G by adding a length-one hint h to H for every join $\{o, p\} \in J$ such that h consists of the edge that connects vertices in $\mathbb{C}_G$ that correspond to the connected components C_o, and C_p.

The graph G, the weight function ω', the maximum weight ω_{max} and the advice H constitute an instance I_{EEA} of EEA.

Theorem 2.3.4. CONJOINING BIPARTITE MATCHING *is polynomial-parameter polynomial-time many-one reducible to* EULERIAN EXTENSION WITH ADVICE *with respect to the parameters join set size and connected components in the input graph.*

Proof. We show that the application of Lemma 2.3.9 and Construction 2.3.3 is such a reduction. It can easily be checked that it can be carried out in polynomial time. Also, by Lemma 2.3.9 and the definition of A_2 it follows that the instances of EEA generated in this way have a number of connected components that is at most the size of the join set plus one.

Assume that there is a perfect conjoining matching M with weight at most ω_{max} for the instance I_{CBM} as in Construction 2.3.3. Then, we derive an Eulerian extension E for G that heeds the advice with the same weight by simply choosing $E := \{(u, v) : u \in I_G^- \wedge \{u, v\} \in M\}$. By the definition of ω', $\omega'(E) = \omega(M)$. Every hint is realized by E because for every join there is an edge in M that satisfies it. Most importantly, E is an Eulerian extension for G: Since M is perfect, every vertex in G has exactly one arc incident in E. Since every vertex in G has balance -1 or 1 (due to the definition of A_1), this suffices to make all vertices balanced. By Lemma 2.3.9(ii), the advice H is a connecting advice and thus $G + E$ is connected.

Now assume that there is an Eulerian extension E for G that heeds the advice H and has weight at most ω_{max}. Choosing $M := \{\{u, v\} : (u, v) \in E\}$ yields a perfect conjoining matching of the same weight: It holds the $\omega'(E) = \omega(M)$, because all extension arcs that do not correspond to an edge in B have weight ∞. The matching M is perfect, because every vertex in I_G^- (in I_G^+) has balance -1 (balance 1), has only incoming (outgoing) allowed arcs and thus has exactly one arc incident in E. The matching M is conjoining, because E heeds the advice H. $\square$

The reduction given above gives rise to the following parameterized equivalence.

Theorem 2.3.5. CONJOINING BIPARTITE MATCHING *and* EULERIAN EXTENSION *are parameterized equivalent with respect to the parameters join set size and connected components in the input graph.*

Proof. By Lemma 2.2.4 there is a parameterized reduction from EE to EECA with respect to the parameter number of connected components. By Theorem 2.3.3 there is a parameterized reduction from EECA to CBM with respect to the parameters connected components and join set size.

The other direction follows from the reduction from CBM to EEA given above in Theorem 2.3.4 with respect to the parameters join set size and connected components and the reduction from EEA to EE given in Theorem 2.2.2. □

We also can finally prove NP-hardness for EECA which we have deferred up to now.

Corollary 2.3.5. EECA *is NP-hard.*

Proof. We have proven in Theorem 2.3.1 that CBM is NP-hard via a reduction from 3SAT. Observe that reducing the instances produced by the corresponding Construction 2.3.1 to instances of EEA by Construction 2.3.3 yields instances with minimal connecting advice. Thus there is a reduction from 3SAT to EECA. □

It turns out that reducing from EECA to CBM and back from CBM to EEA can be interpreted as preprocessing procedure for EECA:

Observation 2.3.6. *Successively applying Construction 2.3.2 and Construction 2.3.3 to an instance of* EECA *yields an equivalent instance of* EECA*.*

Proof. Recall that in Construction 2.3.2 connected components are directly modeled by cells in the vertex partition, hints of length one are directly modeled by joins and hints of length at least two by a gadget comprising of a new cell and two joins, both involving the new cell and one of the endpoints of the hint. Thus, in the corresponding instance of CBM no join can be removed without "disconnecting" one of the cells from the others. Since in Construction 2.3.3 cells are directly modeled by connected components and joins are directly modeled by hints, it follows that the resulting instance has minimal connecting advice. □

This yields the following two results.

Corollary 2.3.6. EECA *has a problem kernel with* $O(b^2c)$ *vertices, where b is the sum of all positive balances and c is the number of connected components.*

Proof. This follows by simply using Construction 2.3.2 and Construction 2.3.3 as preprocessing routines. Since both of them have been proven to be polynomial-time reductions in Theorem 2.3.3 and Theorem 2.3.4 they are correct. Observe that Construction 2.3.2 disposes of all balanced vertices; for every hint of length at least two there are $2b^2$ new vertices, which gives the bound of $O(b^2c)$ vertices in the matching instance. Construction 2.3.3 does not increase the number of vertices and the statement follows. □

Corollary 2.3.7. *For every instance of* EECA *there is an equivalent instance in which every hint has length one.*

Proof. This trivially follows from Observation 2.3.6. □

2.4 Discussion

We now briefly recapitulate the results of this chapter, we note what we could not achieve and we give some directions for further research.

Our considerations in this chapter originally were started with the goal in mind to find out whether EULERIAN EXTENSION (EE) is fixed-parameter tractable with respect to the parameter number c of connected components. Unfortunately, this aim has not been achieved yet. However, we have learned much about the structure of Eulerian extensions in Section 2.1, and could use this knowledge to derive an efficient algorithm for EE in Subsection 2.2.3.

In further research, a useful tool for the analysis of EE with respect to the parameter c could be the parameterized equivalent matching formulation CONJOINING BIPARTITE MATCHING (CBM) we derived in this chapter, the final theorems of which are proven in Subsection 2.3.2. We deem that in this formulation the sought solution is more concisely defined. This observation is partly justified by the work laid out in Sections 2.1 through 2.2.2 in order to catch the structure of Eulerian extensions, which was necessary to finally derive efficient algorithms and arrive at CBM. Also, only considering the structure of the input graph in EE may be misleading since, for instance, balanced vertices also take part in the combinatorial explosion of possible paths in Eulerian extensions. Balanced vertices, however, do not have equivalents in the corresponding matching instance. The matching formulation makes clear that the structure of allowed extension arcs defined by the weight function is of much greater importance. This is also shown in Subsection 2.3.2.2 where we showed that EE is actually tractable with respect to the parameter c for some restricted structure in the allowed arcs. There we used the fact that this structure is precisely captured by the bipartite graph in the matching instance.

Of course we did not stop when we arrived at the matching formulation. We tried multiple approaches for either showing that CBM is fixed-parameter tractable or likely intractable with respect to the parameter join set size. However, this has not been crowned with success yet. For instance, we tried to show W[1]-hardness via parameterized reductions from MULTICOLORED CLIQUE [15] where a graph G, an integer k, and a coloring of the vertices is given and it is asked whether there is a clique K with at least k vertices in G such that each vertex of K has a distinct color. Here it seemed difficult to copy over the information that one vertex is in the clique from one entity representing this vertex to at least k of the vertices' neighbors. Reductions from the well-known INDEPENDENT SET problem suffered from a similar flaw since there it is necessary to copy the information that one vertex is in the independent set over to every neighbor. We also tried reductions from several M[1]-complete problems (see, for instance, Flum and Grohe [16]) without much success.

This led us to the assumption that the bipartite graph in CBM and matchings in this graph are too weak to model the relationships of entities in presumably fixed-parameter intractable problems. Thus, we tried to apply some of the well-known techniques to show fixed-parameter tractability. However, we were not able to circumvent running times in the order of n^j where j is the size of the join set in these approaches. Subsuming, we are not confident with giving a conjecture on whether or not CBM is fixed-parameter tractable.

Chapter 3

Incompressibility

In this chapter we introduce the problem SWITCH SET COVER (SSC) for which there are parameterized reductions to two Eulerian extension problems. We show that polynomial-size kernels for the extension problems would imply polynomial-size kernels for SSC. However, we also show that polynomial-size problem kernels for SSC do not exist unless coNP $\subseteq$ NP/poly.

To prove nonexistence of polynomial-size kernels we use the framework introduced by Bodlaender et al. [3]: An *or-composition algorithm* for a parameterized problem (Q, κ) over the alphabet Σ is an algorithm that

(1) receives a number of instances $I_1, \ldots, I_m \in \Sigma^*$, with

$$\kappa(I_1) = \ldots = \kappa(I_m) = k,$$

(2) runs in time that is polynomial in $\sum_{i=1}^{m} |I_i| + k$, and

(3) outputs an instance $I^* \in \Sigma^*$, such that $\kappa(I^*)$ is bounded by a polynomial in k and $I^* \in Q$ if and only if $I_j \in Q$ for some $1 \leq j \leq m$.

A parameterized problem is called *or-compositional* if there is an or-composition algorithm for it. Using a result by Fortnow and Santhanam [17], it can be shown that if an or-compositional parameterized problem admits a polynomial-size problem kernel, then coNP $\subseteq$ NP/poly [3].

To prove or-compositionality for SSC, we employ a strategy that has been introduced by Dom et al. [9]. The basic idea is as follows: Prove that the problem is fixed-parameter tractable. In the composition algorithm, when there are many input instances, that is, when m above is at least as large as the fixed-parameter running time, solve all the instances using this algorithm and output a trivial yes or no-instance. Otherwise, if there are less input instances, use this fact to create an identification for every instance. These identifications then can be used to create a composition instance that consists of parts which correspond to the original instance.

3.1 Switch Set Cover

First we define SWITCH SET COVER (SSC) and show that it is NP-complete and fixed-parameter tractable. For convenience, we use the following notation.

Definition 3.1.1. Let C be a set of colors. A *C-position* is a multiset with the elements drawn from C. A *C-switch* is a multiset with the elements drawn from

all C-positions. When the color set is clear from the context, we simply speak of positions and switches.

> SWITCH SET COVER
> *Input:* A set C of c colors and k switches each containing a number of positions.
> *Question:* Is it possible to choose exactly one position in each switch such that each color in C is contained in at least one of the chosen positions?

Example 3.1.1. Intuitively one may think of SSC as the following problem: Given a number of light bulbs, each with a unique color, and a number of switches. The switches can be positioned in exactly one of a number of positions specific to the switch. In each position, a switch lights a defined subset of the light bulbs. The question is, given the light bulbs each switch lights in each position, is it possible to choose a position for each switch such that all light bulbs are turned on?

Note that defining switches and positions as multisets instead of plain sets does not add depth to this problem and seems to complicate things at first. However, it simplifies constructions and makes them more convenient to read later on.

Lemma 3.1.1. SWITCH SET COVER *is NP-complete.*

Proof. We first show membership in NP: An example of a certificate for a yes-instance are the chosen position in each of the switches. This certificate is of polynomial size in the input length and, thus, SSC belongs to NP.

NP-hardness of SSC can be seen via a simple reduction from the SET COVER problem. SET COVER has been proven to be NP-hard by Karp [23]. In SET COVER a set S, a family F of subsets of S, and an integer k is given. It is asked whether there is a subfamily $F' \subseteq F$ such that $|F'| \leq k$ and the union of all sets in F' equals S. To solve SET COVER with SSC, introduce a color set C such that there is a bijection between S and C and introduce a switch K. For every set $f \in F$ add a position to K that contains the colors corresponding to elements of f. Then k copies of K form our sought instance of SSC. This instance is polynomial-time constructible because $k \leq |F|$. If there is a solution to the SET COVER instance, we may just choose positions accordingly in the SSC instance and vice versa. $\square$

Lemma 3.1.2. SWITCH SET COVER *can be solved in time* $O^*(2^{ck})$.

Proof. An algorithm to solve SSC may simply try each combination of positions for all the switches: We may assume that in every switch there are at most 2^c positions because positions containing the same colors as other positions may be deleted and multiple copies of one color in one position may also be deleted. Thus, there are at most $(2^c)^k$ combinations of positions. $\square$

3.2 Switch Set Cover is Or-Compositional

We now consider SSC parameterized by the number of colors c and the number of switches k. In order to prove that SSC is or-compositional, we have to give an algorithm as described at the beginning of this chapter. However, if such

an algorithm receives 2^{ck} instances as input, it may directly solve all of the instances and return a trivial yes- or no-instance: Let m be the number of input instances. If $m \geq 2^{ck}$, solving every instance using the algorithm from Lemma 3.1.2 takes $O^*(m2^{ck})$ time. This is polynomial in m. Thus, in the following, we may assume the number m of instances to be smaller than 2^{ck}, implying that $\log(m) \leq ck$. This relation allows us to generate an identification for instances.

Construction Outline. The basic idea is to create an instance-chooser by introducing new switches and colors. Every possible way to choose positions in these new switches shall correspond to exactly one original instance that then has to be solved. In order to achieve this, the input instances are merged by creating new switches that contain all the positions of exactly one switch of every instance. The new colors are then distributed among these merged switches in order to force every solution for the composite instance to solve the chosen original instance.

Composition Algorithm. Let I_i, $0 \leq i \leq m-1$, be instances of SSC, each with c colors and k switches $K_1^i, \ldots, K_k^i$. For convenience and without loss of generality, we assume that each instance uses the same color-set C. Our composition algorithm for SSC works as follows: For each $1 \leq \alpha \leq k$ and each $1 \leq \beta \leq \log(m)$, introduce two colors $o_{\alpha,\beta}^0$ and $o_{\alpha,\beta}^1$. Then, for each $1 \leq \alpha \leq k$, each $1 \leq \beta \leq \log(m)$, and for each instance I_i, if the binary encoding of i has a one at the β'th binary place,[1] add the color $o_{\alpha,\beta}^1$ to every position in the switch K_α^i, otherwise add the color $o_{\alpha,\beta}^0$ to every position in switch K_α^i. Then, create a new instance I^* by creating switches $K_\alpha^*, 1 \leq \alpha \leq k$, where K_α^* contains each of the modified positions of the switches $K_\alpha^i, 0 \leq i \leq m-1$. Finally, introduce switches K_β, $1 \leq \beta \leq \log(m)$, into the instance I^*, where K_β contains one position with the colors $o_{1,\beta}^0, \ldots, o_{k,\beta}^0$ and one position with the colors $o_{1,\beta}^1, \ldots, o_{k,\beta}^1$ and return I^*. See also the pseudocode in Algorithm CompositeSSC and an example of a composite instance in Figure 3.1.

Lemma 3.2.1. *The following statements hold for the new instance I^*:*

(i) I^* has at most $k + ck$ switches and at most $k + 2ck^2$ colors.

(ii) I^* is computable in time polynomial in the sum of the sizes of the input instances.

(iii) I^* is a yes-instance if and only if there is a yes-instance $I_i, 1 \leq i \leq m$.

Proof. Concerning statement (i): There are k switches K_α^* and $\log(m)$ switches K_β in I^*. As we observed at the beginning of this section, $\log(m) \leq ck$. The color-set of I^* consists of k colors from the input instances plus $2k \log(m) \leq 2ck^2$ newly introduced colors (line 7 and 8 in Algorithm CompositeSSC).

Statement (ii) can easily be checked by looking at Algorithm CompositeSSC.

For statement (iii), first assume that there is a yes-instance I_j among the input instances. Then, all c colors in C can be covered by choosing positions in switches of I_j. Since each position of the switches $K_\alpha^j, 1 \leq \alpha \leq k$, is extended (lines 2 to 6) and added to the switches K_α^* (lines 9 to 11), we can choose the corresponding modified positions in each K_α^* to cover the colors in C and the

---

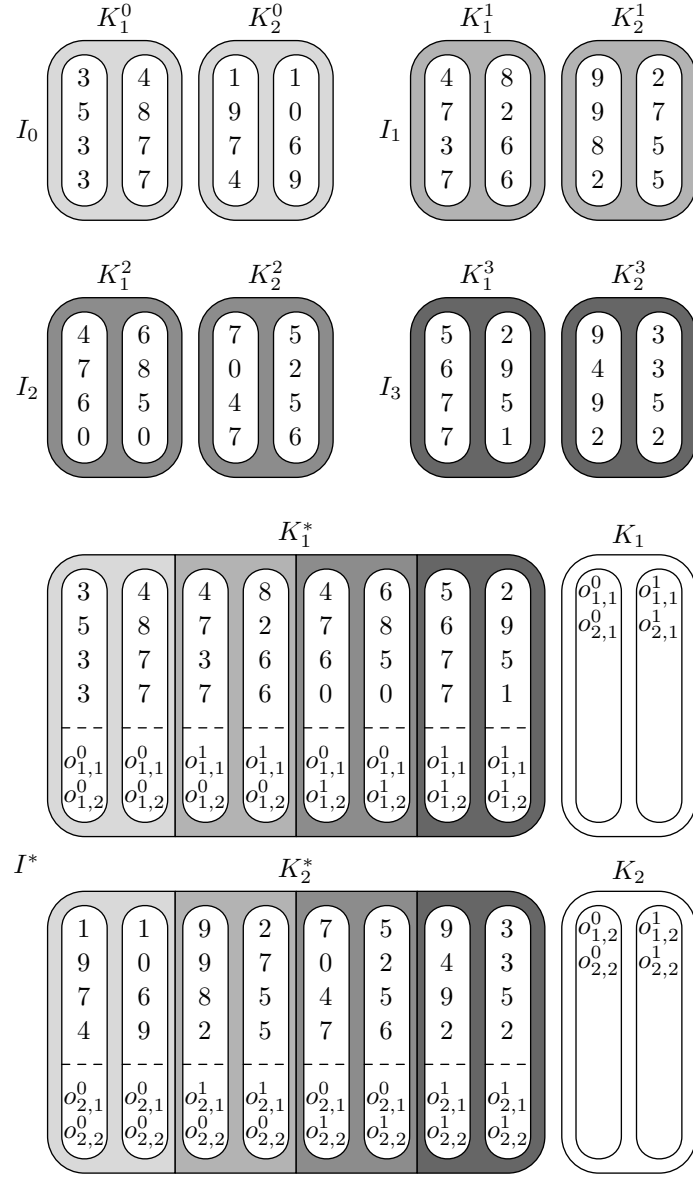[1] Counting the binary places from the right and starting with 1.

Figure 3.1: Four instances $I_0, \ldots, I_3$ of SSC and a composite instance I^* produced by Algorithm CompositeSSC. Each of the instances $I_0, \ldots, I_3$ contains two switches each with two positions. In the composite instance I^* the switches $K_1^0, \ldots, K_1^3$ and the switches $K_2^0, \ldots, K_2^3$ are merged and their positions extended with new colors (positions shaded according to their original instance). Also, in the composite instance, new switches K_1, K_2 are introduced that contain only positions with new colors. If there is a solution to either of the input instances, then we can choose the corresponding positions in I^* and cover the remaining new colors via a position in K_1 and K_2, respectively. Also, if there is a solution to I^*, it has to choose one position in K_1 and one in K_2. The remaining new colors have to be covered by the positions in K_1^*, K_2^*. The only way to cover the new colors is to choose positions in K_1^*, K_2^* that correspond to exactly one of the input instances.

Algorithm CompositeSSC: Composition algorithm for SSC.

    Input: Instances I_i, $0 \leq i \leq m-1$, of SSC, each with c colors from the set C and k switches $K_1^i, \ldots, K_k^i$.

    Output: A composite instance I^*.

1 **for** $1 \leq \alpha \leq k, 1 \leq \beta \leq \log(m)$ **do** generate two new colors $o_{\alpha,\beta}^0$ and $o_{\alpha,\beta}^1$;

2 **for** $0 \leq i \leq m-1, 1 \leq \alpha \leq k, 1 \leq \beta \leq \log(m)$ **do**

3   **for** *each position L in K_α^i* **do**

4   **if** *the binary encoding of i has a one at place β* **then**

5   add the color $o_{\alpha,\beta}^1$ to L;

6   **else** add the color $o_{\alpha,\beta}^0$ to L;

7 $C' \leftarrow C \uplus \{o_{\alpha,\beta}^1, o_{\alpha,\beta}^0 : 1 \leq \alpha \leq k, 1 \leq \beta \leq \log(m)\}$;

8 $I^* \leftarrow$ empty SSC instance with colors C';

9 **for** $1 \leq \alpha \leq k$ **do**

10 $K_\alpha^* \leftarrow$ switch with all positions in the switches $K_\alpha^j, 0 \leq j \leq m-1$;

11   Add K_α^* to I^*;

12 **for** $1 \leq \beta \leq \log(m)$ **do**

13 $K_\beta \leftarrow$ switch with the position $\{o_{1,\beta}^0, \ldots, o_{k,\beta}^0\}$ and the position $\{o_{1,\beta}^1, \ldots, o_{k,\beta}^1\}$;

14   Add K_β to I^*;

15 **return** I^*;

---

colors $\{o_{\alpha,\beta}^{\mathrm{binary}(j,\beta)} : 1 \leq \alpha \leq k, 1 \leq \beta \leq \log(m)\}$, where $\mathrm{binary}(j,\beta)$ denotes the digit of the binary encoding of j at the position β. It remains to cover the colors $\{o_{\alpha,\beta}^{1-\mathrm{binary}(j,\beta)} : 1 \leq \alpha \leq k, 1 \leq \beta \leq \log(m)\}$. This can be done by choosing the positions of the form $\{o_{1,\beta}^{1-\mathrm{binary}(j,\beta)}, \ldots, o_{k,\beta}^{1-\mathrm{binary}(j,\beta)}\}$ in the switches $K_\beta, 1 \leq \beta \leq \log(m)$.

Now, assume that I^* is a yes-instance, that is, assume that it is possible to choose exactly one position in each of the switches of I^* in order to cover all colors of I^*. This implies that there is an integer $0 \leq j \leq m-1$ such that the positions chosen in the switches K_β, $1 \leq \beta \leq \log(m)$, are of the form $\{o_{1,\beta}^{1-\mathrm{binary}(j,\beta)}, \ldots, o_{k,\beta}^{1-\mathrm{binary}(j,\beta)}\}$. None of these positions cover any color of C or $\{o_{\alpha,\beta}^{\mathrm{binary}(j,\beta)} : 1 \leq \alpha \leq k, 1 \leq \beta \leq \log(m)\}$. By construction the colors $o_{\alpha,\beta}^{\mathrm{binary}(j,\beta)}$, $1 \leq \beta \leq \log(m)$, for some fixed α occur only in the switch K_α^*. Furthermore, these colors occur together (that is in one position) in this switch only in the positions that were taken from the instance I_j. In order to cover these colors, one of the modified positions of the instance I_j has to be chosen in K_α^*. This holds for all switches K_α^*, $1 \leq \alpha \leq k$, and since all colors of C are covered, I_j must be a yes-instance. $\square$

Lemma 3.2.1 shows that Algorithm CompositeSSC is a composition algorithm for SSC. Thus, the following theorem follows:

Theorem 3.2.1. SWITCH SET COVER *is or-compositional.*

3.3 Lower Bounds for Problem Kernels

Using the knowledge we have gained about Switch Set Cover (SSC), we can give lower bounds on kernel sizes for Eulerian extension problems. We do this by giving a polynomial-parameter polynomial-time reduction from SSC (parameterized by the number of colors c and the number of switches k) to 2-Dimensional Eulerian Extension (parameterized by the maximum number of extension arcs). Then, since both problems are NP-complete, a problem kernel of polynomial size for 2-Dimensional Eulerian Extension (2DEE) would imply a polynomial problem kernel for SSC—we could simply transform an SSC instance to a 2DEE instance via the parameterized reduction, kernelize it, and then back-transform the underlying non-parameterized 2DEE instance to an SSC instance with polynomial blow-up since the reduction is polynomial-time computable. Furthermore, because 2DEE is a special case of EE (see Subsection 1.2.3), we also obtain lower bounds on the kernel sizes for this more general problem.

In this section, we use the symbols $\prec, \preceq, \succ, \succeq$ for pairs of tuples as "component-wise $<, \leq, >, \geq$", respectively. We also frequently use the notion of allowed arcs. For their definition, see Subsection 1.2.3.

Reduction Outline. The reduction uses the fact that the input graph of an Eulerian extension problem has to be connected by adding extension arcs. Thus, we model colors of an SSC instance as connected components that have to be connected by specific paths consisting of allowed extension arcs. These paths will correspond to the positions in the SSC instance. The main tool we use for the construction are "confined regions" in which vertices can only be connected to one another via extension arcs inside the region and not to vertices outside of the region.

We continue with an intuitive description of our reduction and give a more detailed one in Construction 3.3.1. For the detailed description we need some minor problem restrictions. The descriptions are followed up by an example and after this we give the correctness proof.

Intuitive Description. The idea behind our construction is as follows: It first creates pairs v_i^1, v_i^2 of unbalanced vertices for every switch K_i in the given instance I_{SSC} of SSC. These pairs are interconnected via arcs from the set A_1 that form a cycle such that all pairs belong to one single component. Next, for every position L_j^i in the switches, vertices $w_{i,j,m}$ are introduced that correspond to the colors $1 \leq m \leq c$ in the position. The vertices are placed such that any incoming extension arcs can only originate from one of the vertices of the same position or from the unbalanced vertex corresponding to the switch the position is contained in. Analogously, outgoing extension arcs can only target vertices of the same position or the corresponding unbalanced vertex. Finally, all vertices that correspond to a specific color are interconnected via a directed cycle to create one connected component consisting of balanced vertices for every color. Carrying out these steps, we obtain an instance I_{2DEE} of 2DEE.

In a valid Eulerian extension for I_{2DEE} all connected components of the input graph are connected to one another via extension arcs. Observe that in 2DEE there are no cycles in any valid Eulerian extension because a cycle

must include at least one arc that points upwards-right. Thus, the connected components in I_{2DEE} have to be connected via paths. The placement of the vertices ensures that each such path starts and terminates in the unbalanced vertices corresponding to a single switch and furthermore any such path traverses only vertices corresponding to a single position. Also, the placement of the vertices v_i^1 will ensure that there is no allowed incoming extension arc and thus every Eulerian extension can contain at most one path between v_i^1 and v_i^2. This gives a one-to-one correspondence between SSC solutions that cover all colors and Eulerian extensions in 2DEE that connect all components.

Problem Restrictions. For instances of SSC with color sets of cardinality c we assume that there are exactly c colors in each position—this is without loss of generality, because if there are more colors, then we can delete a repeated color; if there are less, then we can repeat an arbitrary color already in the list. We also assume that the number of positions in a switch is the same for all switches—we can do this because if there is a switch with less positions than in another switch, we can just repeat a position already present.

Construction 3.3.1. Let the color set $\{o_1, \ldots, o_c\}$ and the switches K_i, $1 \le i \le k$, each with l positions L_j^i, $1 \le j \le l$, constitute an instance I_{SSC} of SSC. Construct an instance of 2DEE as follows:

Define the following vertices:

$$v_i^1 := (8cil, 8c(k - i + 1)l) \qquad\qquad v_i^2 := v_i^1 - (4cl, 4cl)$$

Introduce the vertex set $V := \{v_i^1, v_i^2 : 1 \le i \le k\} \cup \{v_0^1, v_{k+1}^2\}$. Connect these vertices using the following arc sets:

$$A_1 := \{(v_{i-1}^1, v_i^1), (v_{i+1}^2, v_i^2) : 1 \le i \le k\} \cup \{(v_k^1, v_{k+1}^2), (v_1^2, v_0^1)\}$$
$$A_2 := \{(v_i^2, v_i^1) : 1 \le i \le k\}$$

Furthermore, for every $1 \le i \le k, 1 \le j \le l, 1 \le m \le c$, define the following vertices:

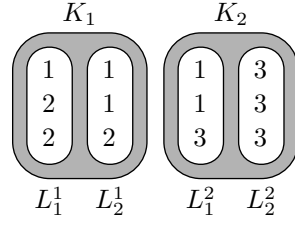$$w_{i,j,m} := v_i^2 + (0, 4cl) + (2c(2j - 1) - 2(m - 1), -2c(2j - 2) - 2(m - 1))$$

For every position L_j^i, let $o_1^{i,j}, \ldots, o_c^{i,j}$ be the colors L_j^i contains and introduce the vertex set $\{w_{i,j,m} : 1 \le m \le c\}$. Let $W_n := \{w_{i,j,m} : o_m^{i,j} = o_n\}$ and let $w_n^1, \ldots, w_n^p$ be a total ordering of W_n. For every $1 \le n \le c$ introduce the following arc set:

$$B_n := \{(w_n^i, w_n^{i+1}) : 1 \le i \le p - 1\} \cup \{(w_n^p, w_n^1)\}$$
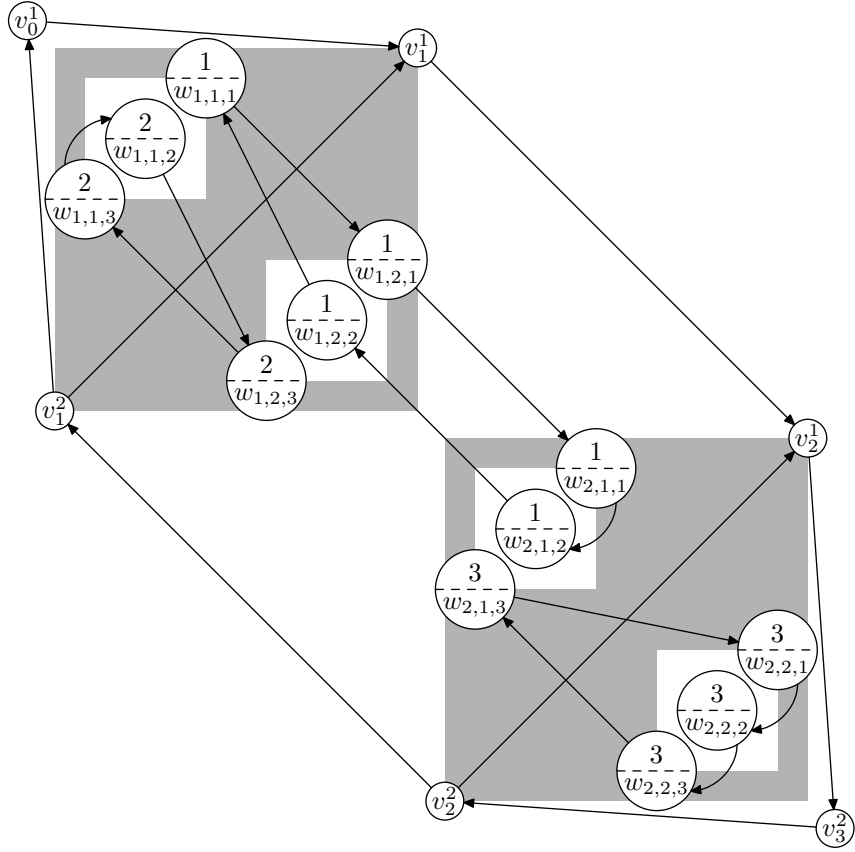
The graph $G := (V \cup \bigcup_{n=1}^c W_n, A_1 \cup A_2 \cup \bigcup_{n=1}^c B_n)$ and the integer $(c + 1)k$ constitute an instance I_{2DEE} of 2DEE.

Example 3.3.1. Consider Figure 3.2. In Figure 3.2a, an instance I_{SSC} of SSC is shown. It contains two switches K_1, K_2 each with two positions. Below this, you can see an instance I_{2DEE} of 2DEE produced from I_{SSC} by Construction 3.3.1— that is, a directed graph embedded in two-dimensional space.[2] It comprises a

---

[2]Not to scale, the coordinates used in Construction 3.3.1 are simplified for readability.

K_1

K_2

1	1
2	1
2	2

1	3
1	3
3	3

L_1^1 L_2^1 L_1^2 L_2^2

(a) SSC instance

v_0^1

$\dfrac{1}{w_{1,1,1}}$

$\dfrac{2}{w_{1,1,2}}$

$\dfrac{2}{w_{1,1,3}}$

v_1^1

$\dfrac{1}{w_{1,2,1}}$

$\dfrac{1}{w_{1,2,2}}$

$\dfrac{2}{w_{1,2,3}}$

v_1^2

$\dfrac{1}{w_{2,1,1}}$

$\dfrac{1}{w_{2,1,2}}$

$\dfrac{3}{w_{2,1,3}}$

v_2^1

$\dfrac{3}{w_{2,2,1}}$

$\dfrac{3}{w_{2,2,2}}$

$\dfrac{3}{w_{2,2,3}}$

v_2^2

v_3^2

(b) 2DEE instance

Figure 3.2: Example application of Construction 3.3.1 explained in Example 3.3.1.

number of vertices represented by circles that may be connected by arcs. The big circles represent vertices that correspond to colors. The number of the color is written in the top half and the vertex name in the bottom half of the circle. Additionally, we see rectangles shaded in gray ("switch regions") and white rectangles ("position regions").

For the switch K_1^1 the pair of vertices v_1^1, v_1^2 is introduced in I_{2DEE} and these vertices are made unbalanced via the arc (v_1^2, v_1^1). Analogously this is done for the second switch. Additionally, two helper-vertices v_0^1, v_3^2 are introduced. They simply ensure that the built graph does not include multiple arcs and remains a simple directed graph. Next, for every position, there are vertices corresponding to their colors. For example, the position $L_1^1 = \{1, 2, 2\}$ corresponds to the vertices $w_{1,1,1}, w_{1,1,2}, w_{1,1,3}$—by Construction 3.3.1 this is due to an arbitrary ordering of the colors in the position, but we stick to the top-to-bottom ordering given by Figure 3.2a here. All vertices that correspond to one single color are connected by a directed cycle. For instance this is the case for the vertices $w_{1,1,2}, w_{1,1,3}, w_{1,2,3}$ which correspond to the color 2.

Consider the solution to I_{SSC} that chooses the positions L_1^1, L_1^2. To solve I_{2DEE} we can just add the directed paths from v_1^1 to v_1^2 and from v_2^1 to v_2^2 also traversing the vertices corresponding to the positions L_1^1, L_1^2, respectively. Also, since the only allowed arcs in 2DEE point downwards-left, any solution to I_{2DEE} consists of paths from the switch vertices v_i^1 to the vertices v_i^2 that traverse other vertices that correspond to exactly one position. Also, in every switch-region at most one path can be in an Eulerian extension, because the vertices v_1^1, v_2^1 have to be balanced and there are no allowed incoming extension arcs. Since the paths connect all connected components of I_{2DEE}, the corresponding positions contain all colors of I_{SSC}. Hence, these positions form a solution to I_{2DEE}.

Correctness. In order to prove the soundness of the construction, we first make some observations about the placement of the vertices. Then, a useful implication of this placement is observed. We then proceed to show that the above-mentioned paths between the unbalanced vertices v_i^1 and v_i^2 are the only allowed arcs in valid Eulerian extensions and, using this, we derive the soundness. The following two observations formulate the notion of regions in the instance.

Observation 3.3.1. *The vertices $w_{i,j,m}$ are contained in the rectangle spanned by all points $p \in \mathbb{Q}^2$ with $v_i^2 \preceq p \preceq v_i^1$.*

Proof. Consider the following difference:

$$v_i^1 - w_{i,j,m} = \begin{pmatrix} 4cl - 2c(2j-1) + 2(m-1) \\ 4c(j-1) + 2(m-1) \end{pmatrix}^\top$$

Both coordinates are positive because $1 \leq j \leq l$. Analogously for v_i^2 and $w_{i,j,m}$:

$$w_{i,j,m} - v_i^2 = \begin{pmatrix} 2c(2j-1) - 2(m-1) \\ 4cl - 4c(j-1) - 2(m-1) \end{pmatrix}^\top$$

The first coordinate is positive because $1 \leq j$ and $1 \leq m \leq c$. The second coordinate is positive because $j \leq l$ and $m \leq c$. $\qquad\square$

Observation 3.3.2. *The vertices $w_{i,j,m}$ are contained in the rectangle spanned by all points $p \in \mathbb{Q}^2$ with $w_{i,j,c} \preceq p \preceq w_{i,j,1}$. Moreover, $w_{i,j,m} \succeq w_{i,j,m'}$ for $m' > m$.*
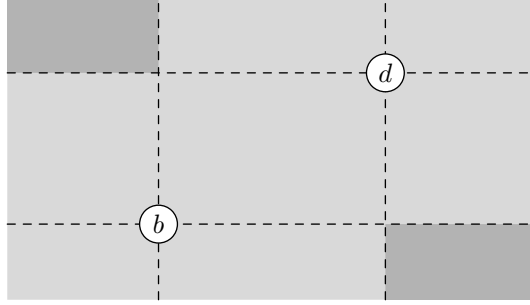
Figure 3.3: The possible placements for tuples $y \in \mathbb{Q}^2, y \preceq c \vee c \preceq y$ for any c such that $b \preceq c \preceq d$ are colored in light grey. This region does not intersect with the region of possible locations for x colored in dark grey (not including the dashed lines).

Proof. It is clear that $w_{i,j,m} - w_{i,j,m'}$, where $m' > m$ is positive, because m and m', respectively, have a negative sign in the definition of these vertices. The statement follows because $1 \leq m \leq c$. $\qquad\square$

We call the corresponding rectangle spanned by v_i^1, v_i^2 the *region of switch K_i* and the rectangle spanned by $w_{i,j,1}, w_{i,j,c}$ the *region of position L_j^i*. The intent of this placement is to exploit the following observation.

Observation 3.3.3. *Let $x, b, d \in \mathbb{Q}^2$ with $b \preceq d$ and neither $x \preceq d$ nor $b \preceq x$. Then, for every $b \preceq c \preceq d$ it holds that neither $x \preceq c$ nor $c \preceq x$.*

Observation 3.3.3 directly follows from looking at Figure 3.3.

Lemma 3.3.1. *Any allowed extension arc in the instance I_{SSC} starts in v_i^1 or $w_{i,j,m}$ for some i, j, m and ends in either v_i^2 or $w_{i,j,m'}$ with $1 \leq m < m' \leq c$.*

Proof. In 2DEE arcs (u, v) are allowed extension arcs if $v \preceq u$. Because of Observation 3.3.1 and Observation 3.3.2 we know that the above mentioned arcs are allowed arcs. It remains to show that they are the only allowed arcs. We first look at arcs between vertices that belong to regions of two different switches $K_i, K_{i'}$. For them—by Observation 3.3.3—it suffices to show that for $x \in \{v_i^1, v_i^2\}, y \in \{v_{i'}^1, v_{i'}^2\}$, neither $x \preceq y$ nor $y \preceq x$. This is the case if in $x - y$ one coordinate is positive and one negative:

$$v_i^1 - v_{i'}^1 = v_i^2 - v_{i'}^2 = \begin{pmatrix} 8c(i - i')l \\ -8c(i - i')l \end{pmatrix}^\top$$

$$v_i^1 - v_{i'}^2 = \begin{pmatrix} 8c(i - i' + 1/2)l \\ -8c(i - i' - 1/2)l \end{pmatrix}^\top$$

Concerning arcs between vertices belonging to regions of two positions $L_j^i, L_{j'}^i$: By Observation 3.3.3 it suffices to show that for $x \in \{w_{i,j,1}, w_{i,j,c}\}, y \in \{w_{i,j',1}, w_{i,j',c}\}$ neither $x \preceq y$ nor $y \preceq x$. Again, we look at the differences and observe that one

coordinate is positive and one negative:

$$w_{i,j,1} - w_{i,j',1} = w_{i,j,c} - w_{i,j',c} = \begin{pmatrix} 4c(j-j') \\ -4c(j-j') \end{pmatrix}^{\top}$$

$$w_{i,j,1} - w_{i,j',c} = \begin{pmatrix} 4c(j-j') + 2(c-1) \\ -4c(j-j') + 2(c-1) \end{pmatrix}^{\top} \qquad \square$$

Lemma 3.3.2. *Construction 3.3.1 is a polynomial-parameter polynomial-time many-one reduction.*

Proof. It is clear that the parameter of the target instance is polynomial in the parameter of the original instance. It can also easily be checked that Construction 3.3.1 can be carried out in polynomial time.

Concerning the correctness, first assume that the original instance I_{SSC} is a yes-instance. Thus, there is a sequence of k positions $L_{j_1}^1, \ldots, L_{j_k}^k$ that cover every color in C. For every $L_{j_\alpha}^\alpha$ take the arcs of the path $v_\alpha^1, w_{\alpha,j_\alpha,1}, \ldots, w_{\alpha,j_\alpha,c}, v_\alpha^2$ into the arc set E. The arc set E is a solution to I_{2DEE} because of the following: First, E contains exactly $(c+1)k$ arcs. Second, every arc of the paths is an allowed arc because of Observation 3.3.1 and Observation 3.3.2. Third, since the positions cover all colors, these paths connect all components of the graph G, that is, $V, W_1, \ldots, W_c$. Fourth, since every vertex v_i^1 (every vertex v_i^2) has exactly one arc starting (ending) in it in E, the graph G has no unbalanced vertices when adding the arcs of E.

Now assume that I_{2DEE} is a yes-instance. In any Eulerian extension E for G, every vertex v_i^1 (every vertex v_i^2) has exactly one incident outgoing (incoming) arc, because it has balance 1 (-1) and there are no allowed incoming (outgoing) arcs (Lemma 3.3.1). Because there are no allowed extension arcs between vertices in different switch regions (Lemma 3.3.1) and thus E consists of a series of k maximal-length paths $p_1, \ldots, p_k$. Each path p_i starts in the vertex v_i^1, ends in the vertex v_i^2 and traverses a subset of vertices in the region of exactly one position. This is because in every region of a switch K_i, the only allowed arcs starting in v_i^1 lead to either v_i^2 or a vertex $w_{i,j,m}$ in the region of the position L_j^i and there are no allowed arcs that lead from one position region to another (Lemma 3.3.1). Let $L_{j_1}^1, \ldots, L_{j_k}^2$ be the positions corresponding to the position region the paths $p_1, \ldots, p_k$ traverse vertices in (if p_i traverses no vertices besides v_i^1, v_i^2, let $L_{j_i}^i$ be an arbitrary position in the switch K_i). Choosing these positions covers all colors in I_{SSC} because the paths connect all connected components of the graph G. $\square$

Using this reduction, the following theorems now arise:

Theorem 3.3.1. Switch Set Cover *parameterized by the number of colors and the number of switches is polynomial-time polynomial-parameter reducible to* 2-Dimensional Eulerian Extension *parameterized by the number of extension arcs.*

Theorem 3.3.2. 2-Dimensional Eulerian Extension *parameterized by the maximum number of extension edges in a solution does not have a polynomial problem kernel, unless coNP $\subseteq$ NP/poly and thus PH $= \Sigma_3^p$.*

Using this theorem, we also can easily derive the following corollary.

Corollary 3.3.1. EULERIAN EXTENSION *parameterized by the number of components and/or the sum of all positive balances of vertices does not have a polynomial problem kernel, unless coNP $\subseteq$ NP/poly.*

Proof. This is because the number c of components in the input graph and the sum b of all positive balances is bounded by the maximum number k of extension edges in a solution. Observe that any Eulerian extension E for the input graph has to connect all connected components. Hence, $c - 1 \leq |E|$ and thus $c - 1 \leq k$. Also, E has to balance every vertex and thus has to include d arcs for every vertex of balance d. Thus, $b \leq |E|$ and thus $b \leq k$.

If there were a polynomial problem kernel for either the parameters c, or b or the combined parameter b, c, this would imply a polynomial problem kernel for the parameter k and the statement follows from Theorem 3.3.2. $\qquad \square$

Chapter 4

Conclusion

In this thesis, we have gained insight into the structure of Eulerian extensions in Section 2.1. We benefited from this knowledge in Subsection 2.2.3 in that we were able to give an efficient parameterized algorithm for the problem EULERIAN EXTENSION (EE) with $O(4^{c\log(bc^2)}n^2(b^2 + n\log(n)) + n^2m)$ running time. Here, c is the number of components in the input graph and b is the sum of all positive balances of vertices in the input graph.

We also gave a reformulation of EE parameterized by c in terms of the natural matching problem CONJOINING BIPARTITE MATCHING in Section 2.3. This formulation might help to attack the fixed-parameter tractability of EE with respect to parameter c from a different angle, and we already gave some partial tractability results in Subsection 2.3.2.2.

Finally, we studied polynomial-time preprocessing routines for EE with respect to either of the parameters k, b, and c, and showed that such routines cannot yield a polynomial-size problem kernel unless coNP $\subseteq$ NP/poly. Thus, polynomial-size problem kernels for EE would imply that the polynomial hierarchy collapses to the third level [31].

Outlook. The most interesting open question is whether EE is fixed-parameter tractable with respect to the parameter c. By the equivalence of EE and CONJOINING BIPARTITE MATCHING (CBM) we gave in Section 2.3, this question is equivalent to whether CBM is fixed-parameter tractable with respect to the parameter "join set size". Intuitively, CBM is a very natural problem—modelling for example the job assignment problem, where at least one worker of a particular profession must be assigned to a facility of a specific type. This makes work for CBM particularly interesting. A way to attack CBM could be by delving into the world of hypergraph transversals [26], since CBM can be seen as a colored variant of the hypergraph transversal problem. Another way of gaining a deeper understanding of CBM could be to search for formulations of this problem that show that it is contained in W[t] for some constant t.

We observed that the parameters b and c are upper bounded by the parameter k used by Dorn et al. [10] in their algorithm for EE with running time $O(4^k n^4)$. Their algorithm uses a dynamic programming approach and likely uses exponential space. We think that our algorithm for EE with running time $O(4^{c\log(bc^2)}n^2(b^2 + n\log(n)) + n^2m)$ can be implemented as to use only polynomial space. In this regard, it would be interesting to see how both

algorithms perform on sets of practical instances.

In this thesis, we focussed on directed Eulerian extension problems. However, the undirected variant of EE is also NP-hard—the NP-hardness proof we gave in Subsection 1.2.2 canonically transfers over to the undirected variants. It would be interesting to see whether our fixed-parameter tractability results also carry over to the undirected problem—we conjecture that this is the case. We also think that the equivalence to a matching problem can be shown in a similar fashion to our observations in Section 2.3. However, we think that the corresponding matching formulation will be a non-bipartite version of CBM, intuitively making the undirected variant of EE a harder problem with respect to parameter c.

We also limited ourselves to adding edges in order to make graphs Eulerian in this thesis. However, there are other natural variants, for instance, deleting edges, editing edges—that is deleting or adding edges—, removing vertices, or any combination of those. Mixed graphs, containing both arcs and edges, are also an option.

We did not consider approximation algorithms in this work. However, given the relationships of EE to HAMILTONIAN CYCLE and the RURAL POSTMAN problems, both of which admit constant-factor polynomial-time approximation algorithms in some special cases, it would be interesting to analyze EE in this regard. Then again, the reduction from SWITCH SET COVER to EE we gave in Chapter 3 might refute attempts in this direction, because SWITCH SET COVER is a variant of SET COVER and this problem is likely not constant-factor approximable [25].

We merely touched the topic of constrained Eulerian extensions in Subsection 1.2.3. We would like to remark that this topic might make for an interesting field of research: It is likely well-motivated through practical problems and we expect the stronger structural restrictions to be exploitable for efficient algorithms. Problems there could be augmenting transitive graphs to Eulerian transitive graphs or the like.

Bibliography

[1] S. Arora and B. Barak. *Computational Complexity: A Modern Approach.* Cambridge University Press, New York, NY, USA, 2009. 5

[2] J. Bang-Jensen and G. Gutin. *Digraphs: Theory, Algorithms and Applications.* Springer, second edition, 2008. 7, 10

[3] H. L. Bodlaender, R. G. Downey, M. R. Fellows, and D. Hermelin. On problems without polynomial kernels. *J. Comput. System Sci.*, 75(8):423–434, 2009. 59

[4] F. T. Boesch, C. Suffel, and R. Tindell. The spanning subgraphs of Eulerian graphs. *J. Graph Theory*, 1:79–84, 1977. 11

[5] A. Cayley. A theorem on trees. *Quart. J. Math.*, 23:376—378, 1889. 32

[6] J. Chen and J. Meng. On parameterized intractability: Hardness and completeness. *Comput. J.*, 51(1):39–59, 2008. 7

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, 2nd edition, 2001. 5

[8] R. Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics.* Springer, 3rd edition, 2005. 7

[9] M. Dom, D. Lokshtanov, and S. Saurabh. Incompressibility through colors and IDs. In *Proc. 36th ICALP*, volume 5555 of *LNCS*, pages 378–389. Springer, 2009. 59

[10] F. Dorn, H. Moser, R. Niedermeier, and M. Weller. Efficient algorithms for Eulerian extension. Manuscript. Preliminary extended abstract appeared in Proc. 36th WG, 2010. 1, 2, 4, 11, 12, 15, 16, 20, 37, 38, 39, 71

[11] R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer, 1999. 6, 7

[12] J. Edmonds and R. M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *J. ACM*, 19(2):248–264, 1972. doi: 10.1145/321694.321699. 43, 45

[13] H. A. Eiselt, M. Gendreau, and G. Laporte. Arc routing problems, part II: The rural postman problem. *Oper. Res.*, 43(3):399–414, 1995. 4

[14] L. Euler. Solutio problematis ad geometriam situs pertinentis. *Comment. Acad. Sci. Imp. Petropol.*, 8:128–140, 1736. 4

[15] M. R. Fellows, D. Hermelin, F. A. Rosamond, and S. Vialette. On the parameterized complexity of multiple-interval graph problems. *Theor. Comput. Sci.*, 410(1):53–61, 2009. 58

[16] J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer, 2006. 6, 7, 58

[17] L. Fortnow and R. Santhanam. Infeasibility of instance compression and succinct PCPs for NP. *J. Comput. System Sci.*, 77(1):91–106, 2011. 59

[18] G. N. Frederickson. *Approximation algorithms for NP-hard routing problems*. PhD thesis, Faculty of the Graduate School of the University of Maryland, 1977. 13, 16, 37

[19] G. N. Frederickson. Approximation algorithms for some postman problems. *J. ACM*, 26(3):538–554, 1979. 16

[20] N. G. Hall and C. Sriskandarajah. A survey of machine scheduling problems with blocking and no-wait in process. *Oper. Res.*, 44(3):510–525, 1996. 4

[21] W. Höhn, T. Jacobs, and N. Megow. On Eulerian extension problems and their application to sequencing problems. Technical Report 008, Combinatorial Optimization and Graph Algorithms, TU Berlin, 2009. 4, 15

[22] S. Kapoor and H. Ramesh. Algorithms for enumerating all spanning trees of undirected and weighted graphs. *SIAM J. Comput.*, 24:247–265, April 1995. 32

[23] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972. 14, 40, 60

[24] J. K. Lenstra and A. H. G. R. Kan. On general routing problems. *Networks*, 6(3):273–280, 1976. 14

[25] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. *J. ACM*, 41(5):960–981, 1995. 72

[26] L. Mirsky. *Transversal Theory*. Academic Press, 1971. 71

[27] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006. 6

[28] C. S. Orloff. On general routing problems: Comments. *Networks*, 6(3): 281–284, 1976. 14

[29] D. B. West. *Introduction to Graph Theory*. Prentice Hall, 2nd edition, 2001. 7

[30] R. J. Wilson. An Eulerian trail through Königsberg. *J. Graph Theory*, 10: 265–275, 1986. 4

[31] C. K. Yap. Some consequences of non-uniform conditions on uniform classes. *Theor. Comput. Sci.*, 26(3):287–300, 1983. 71

Selbstständigkeitserklärung

Ich erkläre, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe.

Jena, den 21. Februar 2011,   Manuel Sorge